Enhancing Trust and Resource Allocation in Telecommunications Cloud

Borger Ormiskangas Vigmostad

School of Electrical Engineering

Thesis submitted for examination for the degree of Master of Science in Technology.

Espoo September 2018

Supervisor

Prof. Raimo A. Kantola, Aalto University

Advisor

Dr. Ian Oliver, Nokia Bell Labs

Aalto University
School of Electrical Engineering

Aalto University
School of Electrical
Engineering

Aalto University, P.O. BOX 11000, 00076 AALTO
www.aalto.fi
Abstract of the master's thesis

Author Borger Ormiskangas Vigmostad

Title Enhancing Trust and Resource Allocation in Telecommunications Cloud

Degree programme Master's Programme in Computer, Communication and Information Sciences

Major Communications Engineering **Code of major** ELEC3029

Supervisor Prof. Raimo A. Kantola, Aalto University

Advisor Dr. Ian Oliver, Nokia Bell Labs

Date September 2018 **Number of pages** 74+17 **Language** English

Abstract

Network Functions Virtualization (NFV) has brought the telecommunications industry multiple benefits; however, it has also introduced many new security issues. This thesis tackles security issues related to NFV trust and defines trust as confidence in the integrity of the software and hardware in a system. Existing NFV trust solutions have added trust to the NFV infrastructure with boot time measurements, placement of Virtualized Network Functions (VNFs) on trusted infrastructure and integrity checks of a small set of VNF operations.

This thesis implements the introduced trust elements from existing solutions and proposes several extensions. These extensions enable trust in the NFV management software with run time measurements, introduces a new method for building VNF trust, extends the number of trusted VNF operations and increases the user auditability of trust decisions.

The proposed extensions are designed, implemented and evaluated in a trusted NFV cloud environment. Although the proposed extensions create a more trusted cloud, they come at a steep performance cost to VNF operations. However, the most impacted VNF operations only affect the cloud provider and not the telecommunications consumer.

This thesis offers a valuable contribution to NFV clouds where increased trust is more important than maximized performance or where VNF operations are rarely performed.

Keywords Trusted Computing, Network Function Virtualization, Cloud Computing, Telecommunications

Preface

This thesis would not have been possible without the great support and encouragement of many people around me. I am grateful to you all.

First of all, I would want to extend my deepest gratitude to my advisor, Dr. Ian Oliver, and to my supervisor, Prof. Raimo A. Kantola.

Thank you Dr. Ian Oliver, for providing me with great insight into the research state of mind, for teaching me how to create strong hardware trust in a system and for helping me formulate my ideas precisely. Not only did you aid me in this thesis work, you have also inspired me to pursue a continued research career in cybersecurity.

Thank you Prof. Raimo A. Kantola, for assisting my greatly in the beginning stages of my writing, for giving me freedom to write the thesis independently, for responding quickly to all my questions and for good guidance in the finishing stages of writing.

I would also want to extend my greatest gratitude to Nokia Networks for funding my thesis work and especially thank my managers during this time: Markku Niiranen and Martin Peylo.

The research and writing of my thesis were done as a part of the security research team at Nokia Bell Labs in Espoo, Finland. I extend my sincerest gratitude to the team as a whole and I would like to especially thank Dr. Ian Oliver, Dr. Yoan Miche, Dr. Silke Holtmanns, Aapo Kalliola, Gabriela Limonta, Isha Singh and former team member Sakshyam Panda. You have all given me immense support both academically and personally.

I would also want to express my greatest gratitude to my family and friends. You support me in everything I do and your continued support makes me strong enough to continuously seek new challenges.

Last but not least, I am eternally grateful to my wife, Marjo, and to my daughter, Klara Amanda. You are the biggest joys in my life and I am very lucky to have you both.

Otaniemi, September 2018

Borger Ormiskangas Vigmostad

Contents

List of Figures

List of Tables

Abbreviations

ACM Authenticated Code Module.

AK Attestation Key.

API Application Programming Interface.

BIOS Basic Input/Output System.

CRTM Core Root of Trust for Measurement.

DRTM Dynamic Root of Trust Measurement.

EK Endorsement Key.

EMS Element Management System.

ENISA The European Union Agency for Network and Information Security.

ETSI The European Telecommunications Standards Institute.

IaaS Infrastructure as a Service.

IMA Integrity Measurement Architecture.

Intel CIT Intel Cloud Integrity Technology.

Intel TXT Intel Trusted Execution Technology.

LCP Launch Control Policy.

MANO Management and Orchestration.

NF Network Function.

NFV Network Functions Virtualisation.

NFVI NFV Infrastructure.

NFVIaaS NFV Infrastructure as a Service.

NIST National Institute of Standards and Technology.

OPNFV Open Platform for NFV.

OS Operating System.

OSS/BSS Operations Support Systems and Business Support Systems.

PCR Platform Configuration Register.

PKI Public Key Infrastructure.

RA Remote Attestation.

RabbitMQ Rabbit Message Queue.

REST Representational State Transfer.

SELinux Security-Enhanced Linux.

SRTM Static Root of Trust Measurement.

TCG Trusted Computing Group.

TEE Trusted Execution Environment.

TPM Trusted Platform Module.

TSecO Trusted Security Orchestrator.

VIM Virtualised Infrastructure Manager.

VM Virtual Machine.

VNF Virtualised Network Function.

VNFaaS Virtual Network Function as a Service.

VNFM VNF Manager.

1 Introduction

Telecommunications is transforming, and an increasing amount of services are being moved into the cloud. This transformation has numerous benefits, including scalability, reduced costs and increased service quality [31, 61].

The transformation is enabled by Network Functions Virtualisation (NFV), which is defined by the European Telecommunications Standards Institute (ETSI) [8]. NFV allows network functions (NFs) to be deployed in the form of software, referred to as Virtualised Network Functions (VNFs). These VNFs can be deployed as one or more virtual machines (VMs) and decouple NFs from hardware. This decoupling removes the old reliance on specialized hardware, as VNFs can run on any general purpose server. Removing dependency on specialized NF hardware improves telecommunications capital efficiencies and reduces the number of different hardware architectures [8]. Additionally, the softwarization of NFs enables quicker deployment and increases the flexibility in starting, stopping and moving NFs, while keeping their original functionality [8].

In an NFV cloud, multiple vendors can develop the different cloud components, including hardware resources, VNFs and management software [8]. This brings many benefits to development and deployment; however, it raises some trust issues as the whole hardware and software stack is no longer controlled by a single vendor [32]. Therefore, as VNFs can run mission critical functions, it is important to establish and maintain a high trust level in the underlying infrastructure, the VNFs and in the cloud management.

This thesis defines trust as confidence in the integrity of hardware and software throughout their life cycles. With this definition of trust, this thesis aims to accomplish one of the high level goals of the NFV specification, which is the integrity protection of hardware and software [11].

1.1 Problem Statement

Although integrity protection is considered in the NFV specification [11], it is not fully developed in any existing implementation or research. Integrity protection in NFV is made difficult due to the amount of vendors [32], the number of elements and the dynamic nature of a cloud [7]. To combat these difficulties, integrity protection should be non-tamperable, flexible and auditable.

The most common device for storing non-tamperable measurements in an NFV cloud is the Trusted Platform Module (TPM) [8]. This is a cryptographic coprocessor designed by the Trusted Computing Group (TCG)[1] to be a hardware anchor on

[1] https://trustedcomputinggroup.org/

which secure systems can be built [5]. This device provides safe storage for boot time measurements of components such as the BIOS, bootloader and kernel. TPM boot time measurements in NFV infrastructure have been used in research [35, 48, 52] and in implementations [34, 37, 60]. However, current implementations allow for little flexibility in changing the expected measurement values and has limited user auditability of measurements.

To enable a trusted NFV cloud, the trust built in the underlying infrastructure has to be extended to include NFV cloud management and VNFs. Boot time measurements are not sufficient, as these elements run after boot time. Early attempts have been made to secure the trust of cloud management with run time measurements stored in the TPM [7, 36]; however, these attempts were only theoretical. Extending the trust into VNFs has been done in research [3, 45, 62, 63] and in implementations [34, 37, 60]. Despite the extensive research, implementations of VNF trust cover only workload placement and a very limited set of life cycle operations, such as VNF start up and suspend. Moreover, they offer minimal trust checks of the VNF supply chain and have limited user auditability of both the VNF movement and measurements.

1.2 Contributions

This thesis contributes to trusted NFV by proposing, implementing and evaluating the following set of trust extensions.

- Extend cloud infrastructure measurements into run time and measure the critical files for cloud management.
- Extend VNF life cycle trust to cover VNF image supply chain and a bigger set of VNF operations, including migration.
- Extend the auditability of the NFV cloud by adding an audit trail of measurements and VNF operations.

These extensions are implemented in a state-of-the-art trusted NFV cloud. In addition to the proposed extensions, the implementation includes improvements made in earlier trusted cloud implementations, such as trust policies [60].

The extensions are evaluated for trust added both on an NFV architectural level and for a specific NFV use case. The chosen use case is the ETSI defined NFV Infrastructure as a Service (NFVIaaS) [9].

1.3 Scope

This thesis aims to construct and maintain trust in a NFV cloud. It does not aim to prevent unwanted changes in NFV, instead it detects and handles them. Security issues beyond integrity measurements, such as access control and secure communication, are not in the scope of this thesis.

1.4 Publications

This thesis has contributed to two original research publications.

The first publication created a testbed for trusted telecommunications systems in a safety critical environment [54]. The trusted VNF solution from this thesis was a part of this testbed and research.

The second publication aimed to combine remote attestation and root cause analysis in an NFV cloud [53]. The trusted NFV cloud implemented in this thesis contributed to the NFV cloud testbed and remote attestation experimentation of the publication.

1.5 Structure of the Thesis

Although the thesis is structured linearly, the chapters are connected as pairs. First, Chapters 2 and 3 review existing knowledge. Then, Chapters 4 and 5 present the thesis contributions. Finally, Chapters 6 and 7 evaluate the contributions and conclude the thesis. Figure 1 shows the flow of this thesis. The remainder of this section briefly introduces each chapter.

Chapter 2 presents NFV, reviews the trusted NFVIaaS use case and introduces trusted computing.

Chapter 3 explains the challenges of creating a trusted NFV cloud; describes existing implementations and research; extends trusted NFVIaaS with the existing trust implementations and reveals the limitations in existing trust solutions.

Chapter 4 introduces the proposed trusted cloud extensions and clarifies how these solve the limitations in previous solutions.

Chapter 5 implements a state-of-the-art trusted NFV cloud with the added extensions.

Chapter 6 evaluates the results of the proposed extensions, extends trusted NFVIaaS with the thesis implementation and tests the implementation performance.

Chapter 7 concludes the thesis and proposes future work.

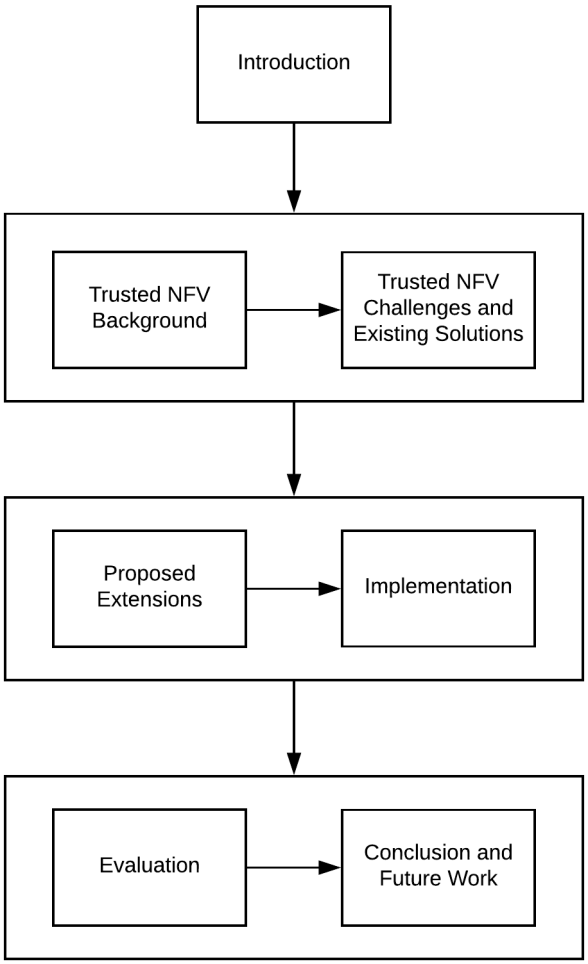

Figure 1: The flow and structure of this thesis

2 Trusted NFV Background

This Chapter presents the technologies needed for creating a trusted NFV cloud and introduces trusted NFVIaaS. First, the Chapter introduces NFV, NFV architecture and most common NFV implementation. Then, it reviews the trusted NFVIaaS use case in a base NFV implementation. Finally, the Chapter presents trusted computing and shows how it can be used to verify the integrity of a platform.

2.1 Network Function Virtualization

Network Functions Virtualisation (NFV) is a network architecture concept that allows for the cloudification of the telecommunications sector by the virtualization of network functions. The NFV architecture is specified by the European Telecommunications Standards Institute (ETSI) and describes the NFV functional blocks and their interfaces [8]. The architecture is on a reference level and does not specify any implementation.

The most common implementation is open platform for NFV (OPNFV) running on OpenStack. This use-case currently implements only a subset of the ETSI NFV architecture. The whole architecture and the OPNFV subset can be seen in Figure 2.

2.1.1 NFV Architecture

The NFV architecture is extensive and meant to cover a wide variety of telecommunications use-cases [10]. The following is a description of the NFV architectural blocks implemented in OPNFV on OpenStack and a brief introduction on the ones either partially or not implemented.

A Virtualised Network Function (VNF) is a virtual network function (NF) that does not depend on specialized hardware. NFs include Evolved Packet Core network elements, such as Mobility Management Entity and Packet Data Network Gateway, and conventional network functions, such as Dynamic Host Configuration Protocol servers. The functional behaviour of a NF is mostly independent of whether the NF is virtualized or traditional. A VNF can be composed of multiple internal components and these components can be deployed over multiple VMs [8].

NFV infrastructure (NFVI) comprises all the hardware and software components on which VNFs are deployed, managed and executed. The infrastructure does not have to be in a single location. NFVI consists of hardware and virtual resources; however, from a VNFs perspective it looks like a single entity providing virtualized resources. The hardware resources of the NFVI provides processing, storage and

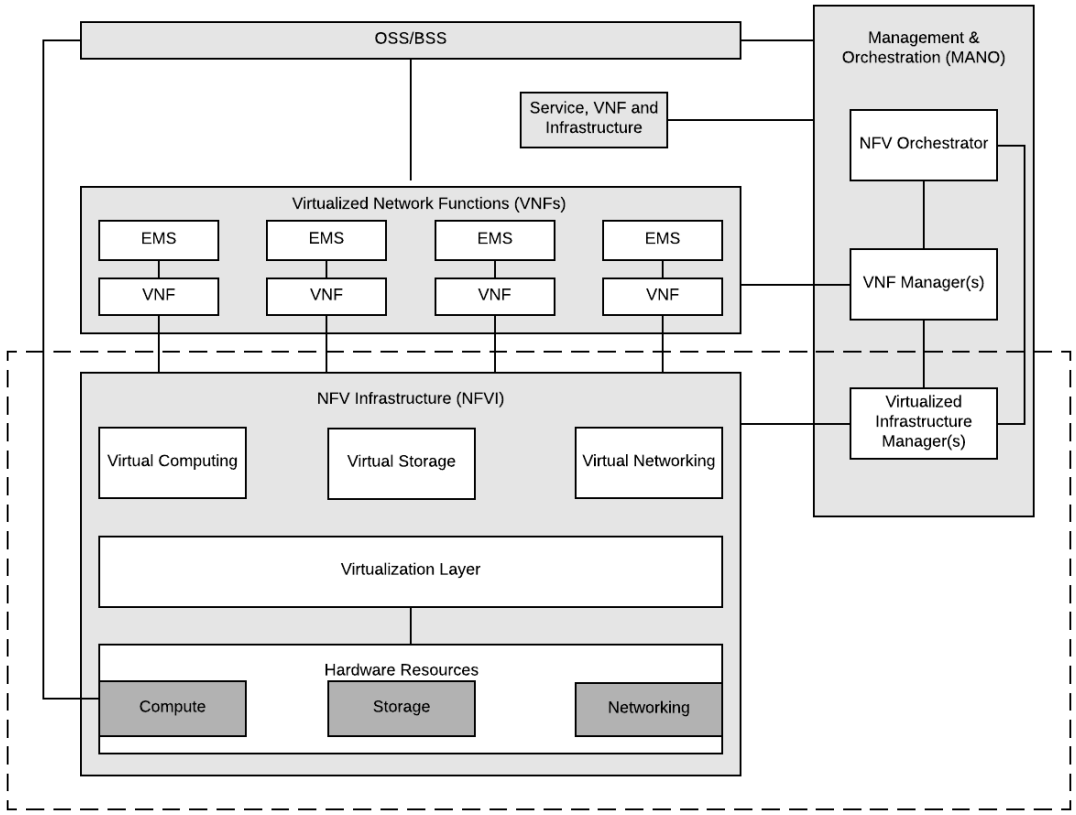

Figure 2: ETSI NFV architecture with OPNFV subset in dotted box [8, 14, 23]

connectivity to VNFs. NFVI hardware is assumed to be commercial-of-the-shelf hardware and not purpose-built hardware. The hardware resources are made available to VNFs through a virtualization layer. This virtualization layer provides VNFs with the virtualized resources they need while abstracting away the hardware logic and location [8].

One or more Virtualised Infrastructure Managers (VIMs) compose the control and management of a VNF's interaction with computing, storage, network and virtualization. A VIM manages the resources of the NFVI and allocates them to VNFs and VMs as needed. Furthermore, it handles the control functions of a NFVI, including the visibility, inventory and fault-handling [8].

Some functional blocks are not completely implemented in OPNFV on OpenStack. These blocks are the Element Management System (EMS), which manages the functionality of one or more VNFs; the Orchestrator, which manages the orchestration of NFVI and software resources; the VNF Managers, which are responsible for VNF life cycle management; Service, VNF and Infrastructure descriptions and Operations Support Systems and Business Support Systems (OSS/BSS) [8].

2.1.2 OPNFV

OPNFV[2] is a Linux Foundation collaborative project created to facilitate the development of NFV. It is currently used in most NFV implementations [14, 23]. Participation in OPNFV is open to anyone and they aim to maintain an open source reference platform [10]. The OPNFV project is also used in commercial products and telecommunications vendors can demonstrate OPNFV readiness and availability by becoming OPNFV verified.

OPNFV depends on multiple upstream projects and does not aim to create OPNFV specific versions of these. These upstream projects include KVM, OpenDaylight, OpenAirInterface and OpenStack. With this philosophy, OPNFV can focus on NFV integration, testing and features built on top of other well established platforms and the interoperability of these platforms [55]. Early OPNFV releases, such as Arno, only support the subset of NFV architecture shown in Figure 2; however, future releases aim to add more upstream projects and cover the entire architecture [55].

2.2 OpenStack as a NFV Platform

OpenStack is a cloud operating system that controls pools of resources in a datacenter, such as computing and networking. These resources are presented through a web interface for management and user provisioning [21]. OpenStack provides OPNFV with the NFVI and Management and Orchestration (MANO) components of the ETSI NFV architecture [14], as marked in Figure 2. In addition, OpenStack can run VNFs as a set of VMs. In this thesis, a one-to-one relation between VNFs and VMs is assumed. This assumption is common in practical deployments [34, 59] and allows OpenStack to take on some of the responsibility of the VNF Manager (VNFM) block in NFV MANO.

Most of the telecom industry state that OpenStack is essential for current implementations of NFV [14, 23]. OPNFV and OpenStack are actively working to make OpenStack compliant to the NFV needs of network operators and new features are added twice per year in new OpenStack releases [14, 17]. This section and this thesis are based on the Pike release of OpenStack.

OpenStack is a collection of numerous service projects with different responsibilities. The Pike release of OpenStack officially supports 36 service projects [21]; however, most of the services are non-essential and installed based on feature needs, such as block storage or Docker support. This allows OpenStack to support a multitude of use-cases.

[2] https://www.opnfv.org/

2.2.1 NFVI

OpenStack provides the infrastructure for NFV, known as the NFVI. The infrastructure comprises compute nodes and a controller node [21]. Compute nodes run VNF workload while the controller runs the MANO functions of the NFV cloud. OpenStack NFVI consists of one or more off-the-shelf servers. The specification of the servers used in this thesis is found in appendix A.

OpenStack NFVI run the OpenStack Nova[3] and Neutron[4] services. Nova virtualizes the hardware resources of the servers and is capable of running VNFs while Neutron provides networking for the servers.

2.2.2 MANO

OpenStack provides OPNFV with Management and Orchestration (MANO) functionality. The OpenStack controller manages its own servers, thus serving as the VIM element of NFV MANO. Furthermore, the controller manages the running VMs, thereby providing parts the VNFM block. OpenStack MANO runs on one of the servers in its NFVI and a node can function as both a compute node and a controller node simultaneously.

OpenStack MANO uses the service projects Keystone[5], Glance[6], Nova, Neutron and Horizon[7]. Keystone provides identity, authentication and service discovery; Glance handles VNF images; Nova manages the scheduling of VNFs; Neutron provides networking and Horizon presents a web management interface.

2.3 OpenStack Internal Communication

Internal OpenStack communication between compute and controller nodes is handled by Rabbit message queue (RabbitMQ)[8]. In RabbitMQ, messages are sent out to various exchanges. Each of these exchanges can have multiple queues, and the queues can be directed at a topic, such as compute, or a topic on a specific host, such as the compute service running on hypervisor number 1. All queues can have any number of consumers, which share the load equally, and all nodes can be both consumer and publisher [33].

Figure 3 shows an example RabbitMQ flow, where the Nova scheduler is sending

[3]https://docs.openstack.org/nova/pike/
[4]https://docs.openstack.org/neutron/pike/
[5]https://docs.openstack.org/keystone/pike/
[6]https://docs.openstack.org/glance/pike/
[7]https://docs.openstack.org/horizon/pike/
[8]https://www.rabbitmq.com/

messages to different Nova compute nodes.

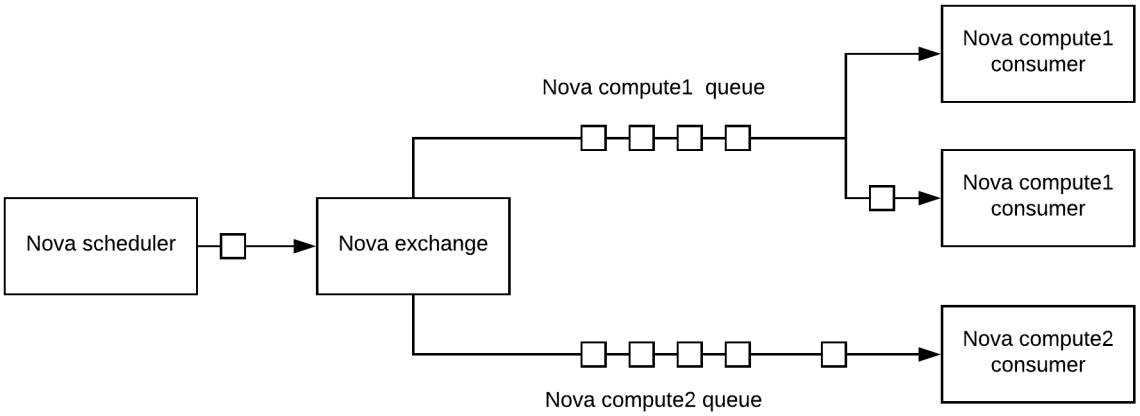

Figure 3: RabbitMQ example flow

2.4 OpenStack VNF Life Cycle

As VNFs do not depend on any specialized hardware, they can run on OpenStack as normal VMs. OpenStack provides an NFV cloud with numerous cloud benefits by allowing the VNFs to be scheduled and operated in a scalable and flexible way [14, 23].

2.4.1 VNF Scheduling

In VNF scheduling, VNFs are placed on an available hypervisor by OpenStack MANO. The hypervisor selection is done by evaluating a set of hypervisor filters that checks for sufficient computational resources, including the number of available CPUs and the amount of available RAM [21]. This placement process is a part of the start VNF operation, which is shown in Figure 4.

2.4.2 VNF Life Cycle Operations

In addition to running VNFs, OpenStack can perform many VNF life cycle operations. These operations change the VNF state. A full VNF state diagram is included in Appendix B Figure B1.

This thesis focuses on three commonly used VNF operations.

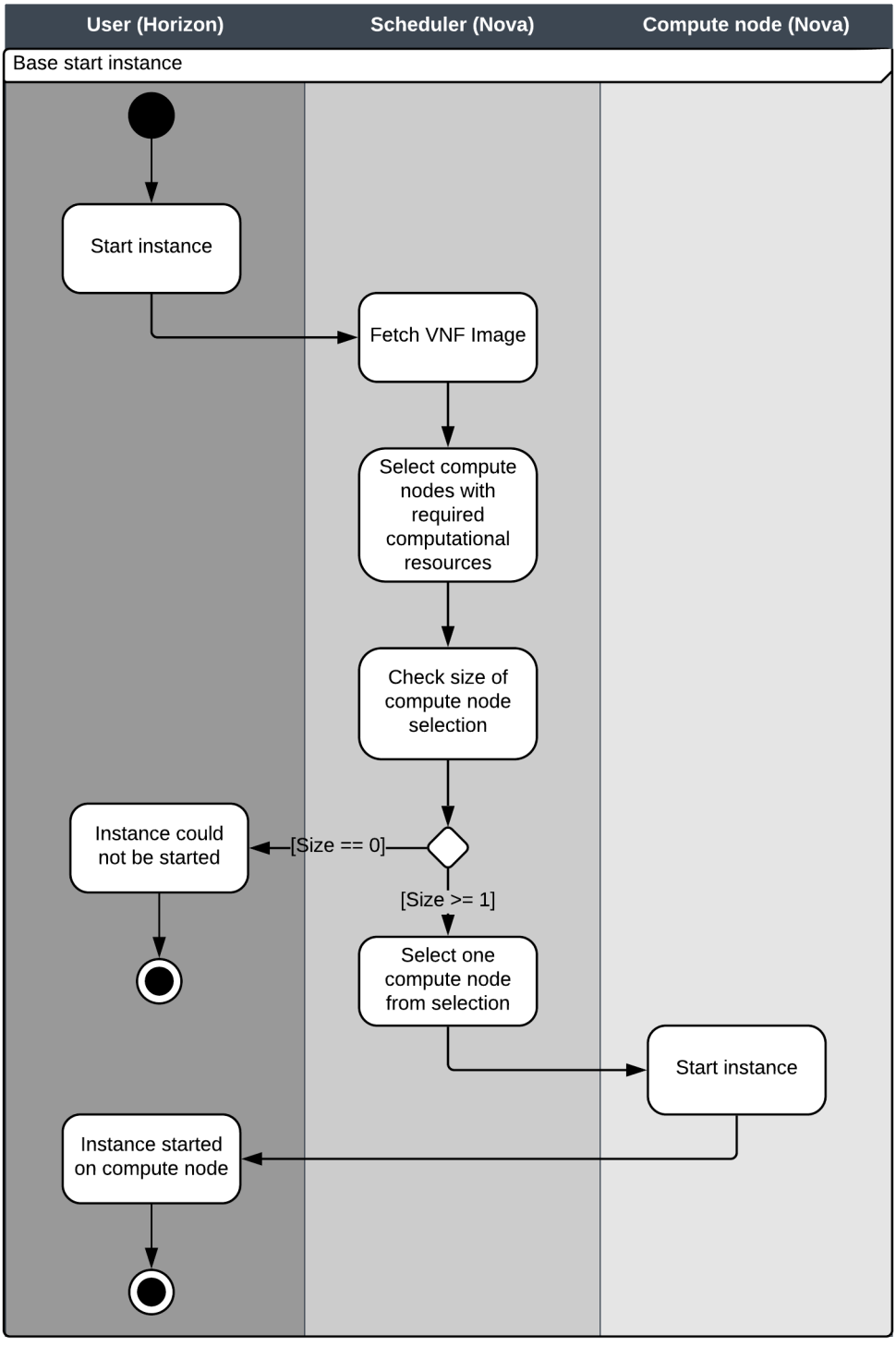

Figure 4: OpenStack start VNF instance

- Suspend, which stops the VNF instance and saves it to memory on the hypervisor.
- Resume, which loads a VNF instance from memory and starts it.
- Migrate, which stops a VNF instance and transfers it to a new hypervisor where it is started.

Figures 5, 6 and 7 show the steps of these three operations in OpenStack.

2.5 OpenStack NFV entities

To perform its part of OPNFV, OpenStack utilizes a set of NFV entities. The entities of an OpenStack OPNFV cloud are hardware elements, VNF images and VNF instances [21].

Hardware elements in OpenStack are commercial-off-the-shelf servers capable of running virtual workload. OpenStack hardware elements function as the NFVI layer of the NFV architecture. Additionally, one hardware element runs the NFV MANO layer. Other NFV hardware elements, such as routers [8], are not implemented in OpenStack nor in this thesis.

In OpenStack, VNF images form one half of the NFV VNF layer. VNF images are in OpenStack VM images. Although VNFs and VMs do not always have a one-to-one relation in NFV [8], it is a common simplification in practical deployments. This simplification is done in OpenStack [21], this thesis and existing trust solutions [34, 59]. VNF therefore equals VM in this thesis and extra VNF data is not considered.

The other half of the OpenStack VNF layer is formed by VNF instances. VNF instances are deployed VNF images and this deployment is done on one of the servers in OpenStack NFVI. VNF instances can perform many operations in OpenStack. The life cycle operations covered in this thesis were described in Section 2.4, while the full VNF state diagram can be found in Appendix B Figure B1.

2.6 Trusted NFV Infrastructure as a Service

NFV Infrastructure as a Service (NFVIaaS) is an ETSI defined NFV use case [9], which in turn is a special case of the National Institute of Standards and Technology (NIST) defined Infrastructure as a Service (IaaS) [44]. While IaaS only needs to provide a pool of resources to a cloud tenant [44], NFVIaaS should also support the operational life cycle of the tenant VNFs [9]. This section will consider trust in NFVIaaS as provided by OpenStack.

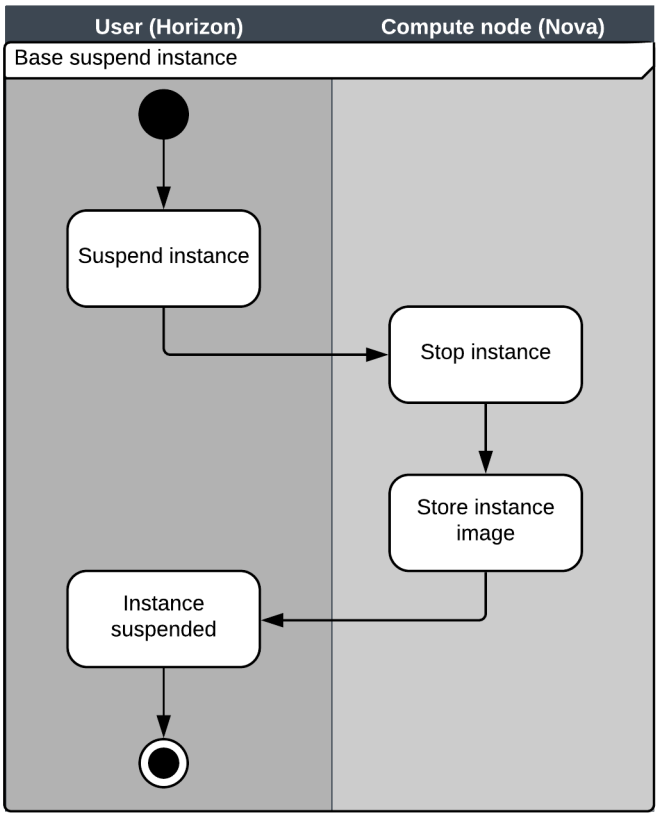

Figure 5: OpenStack suspend VNF instance

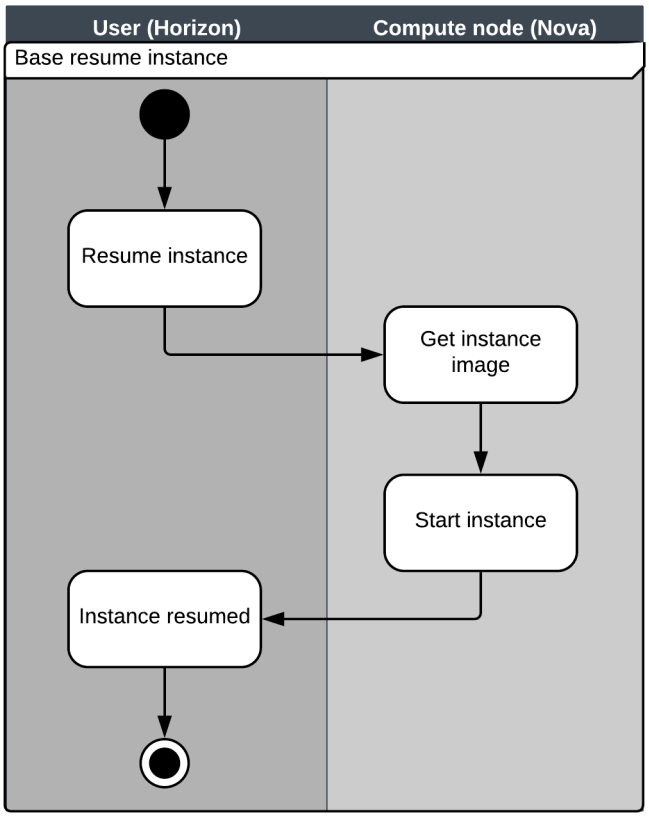

Figure 6: OpenStack resume VNF instance

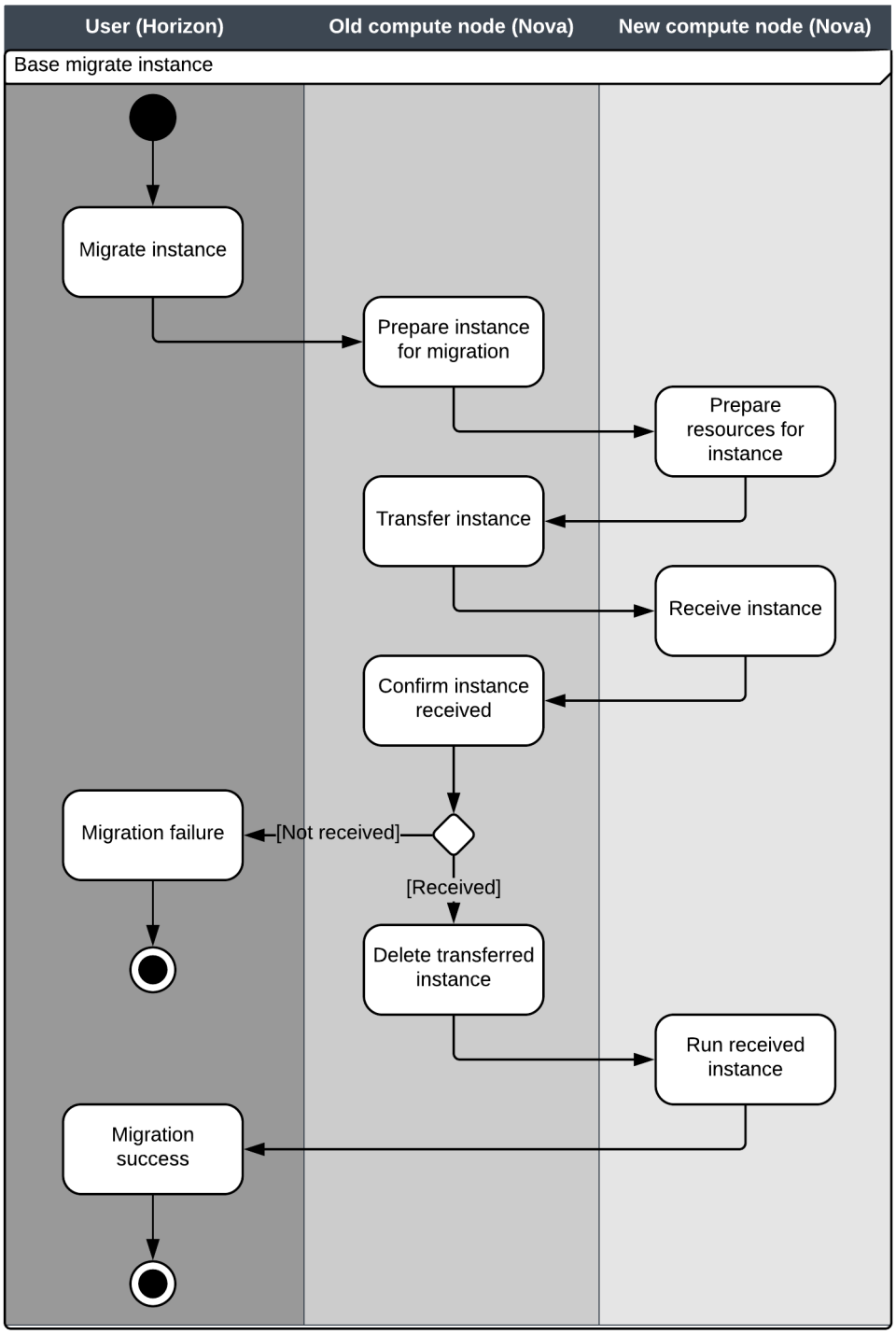

Figure 7: OpenStack migrate VNF instance

2.6.1 Trusted NFVIaaS Success Scenario

The actors in trusted NFVIaaS are the cloud tenant and the service provider. The cloud tenant wants VNF instances that are trusted throughout their life cycles, while the service provider should provide this capability.

The success scenario of trusted NFVIaaS allows the cloud tenant VNF to be trusted in its entire life cycle. All actions by the cloud tenant are done through the OpenStack management software. The steps for the success scenario are listed below.

1. Service provider adds trusted hardware elements to the cloud.
2. Service provider installs a trusted OpenStack configuration.
3. Cloud tenant adds a trusted VNF image.
4. Cloud tenant deploys a VNF image on trusted hardware.
5. Cloud tenant performs trusted VNF operations.
6. Cloud tenant audits the trust decisions.

These steps cover all OpenStack entities and available OpenStack cloud operations. OpenStack entities were described in Section 2.5, while the available OpenStack cloud operations were described in Section 2.2.

The available trust will be examined in this chapter for base OpenStack, in Chapter 3 for the existing OpenStack trust solutions and in Chapter 6 for the solution proposed by this thesis.

2.6.2 Trusted NFVIaaS on Base OpenStack

Base OpenStack does not perform any integrity checks and therefore cannot provide any of the steps needed for trusted NFVIaaS. However, OpenStack does have some recommendations for trust, some trusted features have existed in the past and some will in future releases.

OpenStack recommends ways of adding trust to hardware elements at both boot time [20] and run time [16]. OpenStack has had the notion of trusted hardware elements in the past [22]; however, this functionality has been removed [16]. Signed VNF images are included in the coming release of the OpenStack service project Barbican[9] [18].

Despite existing recommendations, current OpenStack release (Pike) does not provide any trust in NFVIaaS.

[9]https://docs.openstack.org/barbican/

2.7 Trusted Computing

Trusted computing is a set of solutions conforming to the standards and specifications formulated by the Trusted Computing Group (TCG) [10]. TCG technologies aim to secure critical systems, authentication, user identities, machine identities and network integrity. TCG technologies are in use in over a billion devices and TCG has working groups for technologies such as the TPM, trusted network communications and virtualized platforms [29]. This section presents the TCG specified TPM and how it can be used together with trusted execution environments (TEEs) to enable measured boot.

2.7.1 Trusted Platform Module

Trusted Platform Modules (TPMs) are cryptographic coprocessors that are nearly ubiquitous in commercials PCs and servers [5]. They were designed by the TCG to be hardware anchors on which secure systems could be built; therefore, TPMs are physically attached to the motherboard of a PC [5]. This allows computing platforms to verify the software configuration from a place outside of the system memory space [56].

This thesis only consider hardware TPMs on the x86 platform. This is the most widely deployed TPM platform; however, the TCG also has TPM working groups for other platforms, including mobile and virtualized platforms [29]. TPMs first widely deployed version was 1.1b and the most recent version is TPM 2.0. Figure 8 shows the components of a TPM 2.0. The remaining paragraphs of this section summarize some of the main TPM 2.0 features and use cases; however, many of these were similar in earlier TPM versions.

A TPM has at least two pairs of public/private keys available, namely the endorsement key (EK) and the attestation key (AK). The EK is created from an endorsement seed that is added to the TPM at production time. The AK is created from the EK and is intended for signing. These keys are created by the TPM key generation component, while the endorsement seed is stored in TPM non-volatile memory. Both of these components can be seen in Figure 8. Endorsement seed, EK and AK are unique and their private parts never leave the TPM [30].

TPMs have special registers, named platform configuration registers (PCRs), which stores hash values. These registers are stored in volatile memory, as seen in Figure 8. TPM 2.0 supports multiple PCR banks, where each PCR bank is associated with a specific hashing algorithm. Figure 9 shows example PCR values from the SHA-1 PCR bank. PCR values cannot be set directly, instead they are extended from other PCR values. When extending a PCR value, the new value becomes the

[10] https://trustedcomputinggroup.org/

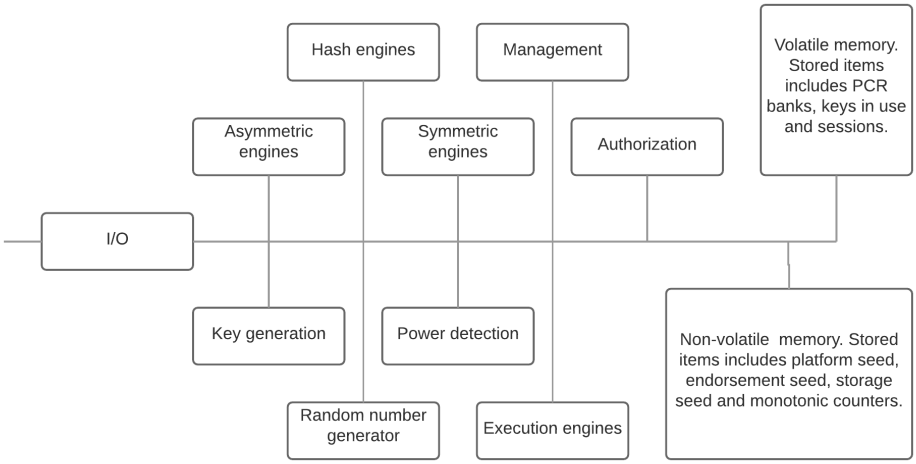

Figure 8: TPM 2.0 block structure [30]

hash of the old PCR value concatenated with the input hash value. Hash functions are assumed to be a one-to-one function and therefore PCRs represents a unique operation sequence. TPM PCRs are used to safely store software measurements and are critical for solutions such as measured boot [56].

A TPM can seal keys or other data to known states of a platform. This means that the availability of these keys or data will depend on platform measurements stored in the TPM. One use case for this is Microsoft's BitLocker [49]. It seals the encryption key in the TPM and links it to PCR values. With this seal, the encryption key can only be accessed if the PCRs are a certain value, i.e., the measured software is in a known state [49].

TPMs can report their software configuration over a network connection by using what is known as a TPM quote. A TPM quote is a signed collection of TPM stored data, including a hash of given PCRs, TPM clock value and a given nonce. Since the TPM signing key, the AK, is unique for each TPM, then a signed TPM quote will be a hardware based assurance of both identity and software configuration.

2.7.2 Trusted Execution Environments

A trusted execution environment (TEE) is an execution environment that is separated from normal processing and provides a higher level of security. TEEs originated as part of an initiative by the TCG. This thesis uses a TEE recommended for use in NFV named Intel Trusted Execution Technology (Intel TXT) [24]; however, other options for TEEs exists, such as AMD Secure Execution Environment[11] and ARM

[11] https://www.amd.com/en/technologies/security

```
sha1 :
  0  : ae62d40b59561f52419a6dbb1fe3f589e7c67856
  1  : b2a83b0ebf2f8374299a5b2bdfc31ea955ad7236
  2  : b2a83b0ebf2f8374299a5b2bdfc31ea955ad7236
  3  : b2a83b0ebf2f8374299a5b2bdfc31ea955ad7236
  4  : b2a83b0ebf2f8374299a5b2bdfc31ea955ad7236
  5  : b2a83b0ebf2f8374299a5b2bdfc31ea955ad7236
  6  : b2a83b0ebf2f8374299a5b2bdfc31ea955ad7236
  7  : 4037336fa7bc0eabe3778fcfff5fcd0ee6adcde3
  8  : 0000000000000000000000000000000000000000
  9  : 0000000000000000000000000000000000000000
  10 : d61c6a58cb72e81f5b5043440d2b50f597d10ffc
  11 : 0000000000000000000000000000000000000000
  12 : 0000000000000000000000000000000000000000
  13 : 0000000000000000000000000000000000000000
  14 : d701bb26d3597931f4b1c6ed5a3cd8d72d15b977
  15 : 0000000000000000000000000000000000000000
  16 : f0e5a9f7d5a33a281fff32e565cb407df5b191c1
  17 : bfb0f7c615ba4f7f76e440202a9701461ad7504c
  18 : 9b8547051d0e68839490e82f99a17d78e3e94fed
  19 : 0000000000000000000000000000000000000000
  20 : 0000000000000000000000000000000000000000
  21 : ffffffffffffffffffffffffffffffffffffffff
  22 : ffffffffffffffffffffffffffffffffffffffff
  23 : 0000000000000000000000000000000000000000
```

Figure 9: SHA-1 PCRs of a TPM

TrustZone[12].

Intel TXT is a set of CPU extensions that can create a measured launch environment [28]. This measured launch environment is used in solutions that provide verified launch, Launch Control Policy (LCP), secret protection and attestation [28]. Verified launch boots a trusted operating system (OS) by doing an accurate and cryptographically verified comparison of critical elements of a launch environment against a known good source. LCP prevents a platform from booting if the boot measurements are not as expected. Secret protection writes secrets to protected memory. Finally, attestation provides the ability to attest the verified launch to local or remote users in a secure manner.

Intel TXT does not function unless multiple other components exists on a platform, including trusted extensions in the processor, authenticated code modules for secure measurements and a TPM for secure storage [28]. Intel TXT does not do the actual measurements itself, instead it enables other applications, such as tboot[13], to measure the kernel safely.

[12]https://www.arm.com/products/security-on-arm/trustzone
[13]https://sourceforge.net/projects/tboot/

2.8 Chain of Trust

Chains of trust extend trust originating from a trust anchor into a platform. The most common chain of trust is used with X.509 architecture, where a root certificate is used as the trust anchor. This thesis constructs a chain of trust based on firmware and software measurements and utilizes the TPM as a trust anchor. This Section explains how to construct a chain of trust using both boot time and run time measurements and the complete chain of trust can be seen in Figure 10

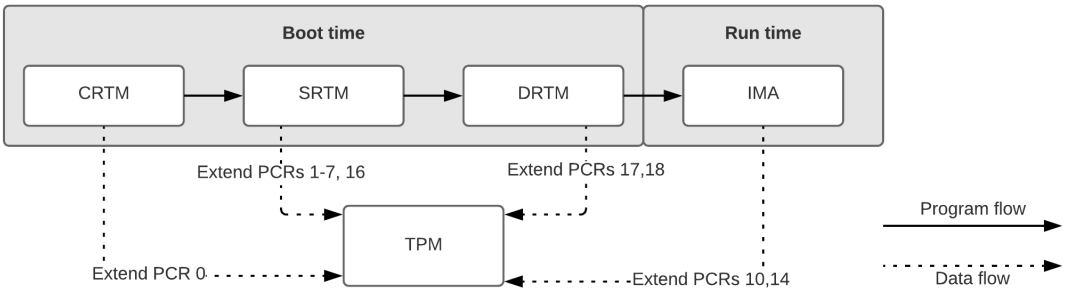

Figure 10: Chain of trust with a TPM trust anchor

2.8.1 Boot Time Measurements

Boot time measurements are measurements done before the OS kernel is running. Typically, code is measured before it is run and all measurements are written into the TPM.

The measurements are divided into multiple sets: the core root of trust for measurement (CRTM), the static root of trust measurement (SRTM) and the dynamic root of trust measurement (DRTM). CRTM is at a well established location in firmware and it is read by the CPU after a power on reset [30]; SRTM includes the measurements of the basic input/output system (BIOS), option ROM and bootloader and DRTM includes measurements of the OS, its kernel modules and early drivers [28, 70].

CRTM and SRTM measurements are supported by TPM drivers, while DRTM requires a TEE and a measurement software, such as TrustedGrub [14] or tboot. All the measurements, both firmware and software, are stored in the TPM PCRs. As PCRs use hash extension, any change in the software, firmware or boot order will be reflected in written PCRs.

[14] https://github.com/Rohde-Schwarz-Cybersecurity/TrustedGRUB2

Implementations of boot time measurements are known as measured boot or trusted boot. These solutions are similar and in some cases identical. Measured boot is a part of the TCG architecture and its aim is to detect any changes in the platforms pre-boot environment [56].

2.8.2 Run Time Measurements

Run time measurements begin after the kernel has loaded and measures files while the system is running. Unlike boot time trust, which can only detect persistent changes [7], a run time trust implementation can track ephemeral changes in files.

Run time measurements can be provided by Linux Integrity Measurement Architecture (IMA)[15]. IMA is an open source trusted computing component comprising several Linux kernel modules with the goal of documenting the integrity of files. It has been included in the Linux kernel since version 2.6.30.

Linux also have other means of run time measurements, including the userspace component auditd [67]. However, only Linux IMA is used in this thesis.

IMA maintains a runtime measurement list of selected files in the filesystem. It can be implemented with or without a hardware anchor, such as a TPM. If IMA is implemented without a hardware anchor, then an adversary can alter the measurement list undetected [43]. A hardware anchor ensures that any change in the files themselves or a change in the measurement list is detected. IMA measurements do not prevent changes to files, as in integrity protection, they only document changes in real time [43]. However, IMA can be extended with IMA-appraisal, which can make files unable to open if the measured values do not match known good values [43]. If implemented with a TPM, then a hash of the measurement list can be written into TPM PCRs.

IMA has quite wide measurement policies and lacks a manner of marking specific files for measurement. An approach to get more granularity in measurement selection is to use Security-Enhanced Linux (SELinux)[16].

SELinux is a Linux security module for improved access control. Base Linux access controls are modifiable by both user and running applications and provide limited security options [47]. SELinux provides enhancement not only in the granularity of access control, but also greater restrictions on who can load access control policies to the system [47]. In addition to security enhancements, SELinux allows a user to tag files. Linux IMA can use these tags for measurements policies. This enables IMA run time measurements of any chosen set of SELinux tagged files.

[15] https://sourceforge.net/p/linux-ima
[16] https://github.com/SELinuxProject

2.9 Secure Boot

Another approach to trust-enhanced boot is Microsoft's secure boot [50], were an approved signer signs critical programs, such as bootloader and OS. When a platform with secure boot starts, these programs are then checked for correct signature using a key stored in firmware. In this way, only signed code is allowed to run on the platform [50]. Secure boot alone is not enough for current trusted NFV implementations nor the solution proposed by this thesis as they all rely on boot-time measurements [34, 59]. However, measured boot and secure boot are complimentary and can be used together.

One security issue with secure boot is that also malware can be signed and cases of this has been found in recent research [38, 39, 40]. With secure boot, signed malware will be verified as trusted and the system will run as normal.

Note that the NFV documentation also operates with a notion of secure boot [11]. NFV secure boot is a version of measured boot and not Microsoft secure boot [50].

2.10 Summary

This Chapter introduced NFV, NFVIaaS and trusted computing. First, the Chapter explored the NFV architecture, how it is most commonly implemented in OpenStack and the entities in OpenStack NFV. Then, the Chapter introduced NFVIaaS and discovered that a base installation of OpenStack currently supplies no trust solution. Finally, the Chapter showed some of the most important elements of trusted computing and how trust can be constructed in a single platform.

The next Chapter explores the challenges in combining NFV and trusted computing and reviews existing solutions from both the NFV architecture view and for the NFVIaaS use case.

3 Trusted NFV Challenges and Existing Research

This Chapter reviews the challenges of implementing a trusted NFV cloud and existing solutions. Although there has been a large amount of research on trusted clouds, there exists only two trusted NFV implementations. This Chapter starts by introducing these two implementations. Then, the Chapter presents the challenges and existing solutions for the NFVI, MANO and VNF blocks respectively. Then, the NFVIaaS use case is extended with existing trust implementations. Finally, the Chapter identifies limitations in existing implementations and research.

3.1 Intel Cloud Integrity Technology

The most used trusted NFV cloud implementation is Intel Cloud Integrity Technology (Intel CIT). Intel CIT is the second generation of Intel's trusted NFV cloud. Their first generation, OpenAttestation[17], was recommended by OpenStack as a way to create trusted compute pools [22]. This recommendation has been removed, and Intel CIT is not officially recommended by either ETSI or OpenStack.

Intel CIT secures cloud workload with workload placement, encryption and launch control [34]. It provides infrastructure boot time trust and extends this into OpenStack and Docker. This summary will only cover the infrastructure and OpenStack parts.

Intel CIT adds multiple elements to the NFV architecture. This Section focuses on the attestation server and the trust agent functionalities, which allows infrastructure trust to be used with OpenStack. Infrastructure trust is built with measured boot and includes measurements of CRTM, SRTM and DRTM. This trust is reported to OpenStack from the trust agent via the attestation server. Infrastructure trust is used to build a trusted pool of hypervisors, on which trusted workload can be placed [34].

Intel CIT allows trusted VNF images to be manually added to the attestation server. A trusted VNF image is in this context selected parts of a VNF image and its expected hash value. VNF image trust is checked during boot for multiple OpenStack VNF operations. The VNF operations supported by Intel CIT are launch from image, suspend, resume, hard reboot, shut off and start [34].

[17] https://01.org/OpenAttestation

3.2 Master Thesis Trust Implementation

In addition to Intel CIT, there is one alternative trust implementation available. This trust implementation was made in a master thesis at Aalto University [59]. The implementation provides infrastructure trust at boot time and extends this to cover workload placement. Furthermore, the implementation checks VNF images through signatures and measurements [59].

This implementation adds some elements to NFV architecture. These elements are an attestation server and the Trusted Security Orchestrator (TSecO). The attestation server is from Intel CIT's solution and the TSecO was developed for this implementation. The attestation server reports integrity measurements of the servers to the TSecO. The TSecO has multiple roles in this implementation, including binding TPM values to a server via the attestation server, signing VNF images through a signing authority and assisting OpenStack in workload placement [59]. In addition to signing VNF images, the integrity of images are checked during workload placement [59].

The infrastructure trust in this trust implementation is constructed similarly to Intel CIT. It builds on measured boot and covers CRTM, SRTM and DRTM. The trust implementation offers different trust-policies based on sets of TPM PCR values. In addition to infrastructure trust, this implementation offers trusted workload placement. This places VNFs on infrastructure matching their trust requirements by using OpenStack filters [59].

3.3 NFVI Trust

Creating and maintaining trust in infrastructure is a hot research topic [32, 71, 73] and the main challenges are in two areas: creating a trusted element and attesting other elements.

There are multiple ways to create a trusted element, such as secure boot [50] and measured boot. Both the existing trusted NFV implementations use measured boot through the trust agent component from Intel CIT [34, 59]. Common approaches in research are measured boot [11] or a combination of measured boot and secure boot [7].

The information from measured boot is commonly distributed through a process of remote attestation (RA). RA can be done with or without a trust anchor. This thesis assumes a TPM trust anchor is available; however, there exists many RA schemes that do not depend on a TPM [1, 51, 64]. In RA with a TPM available on a element, the process is as follows.

- User, verifier or attestation server send a nonce and a list of PCRs to an element

with a TPM.

- The TPM on the element creates a signed TPM quote containing the received nonce and a hash of the given PCRs.
- The sender of the attestation request checks the received TPM signature, nonce and received PCR hash against known good values.

This approach of TPM enabled RA is used in both existing implementations [34, 59] and in research [27, 72, 73].

3.4 MANO Trust

OpenStack implements the MANO functionalities of OPNFV; therefore, trust in OpenStack is equivalent to trust in MANO. To enable trusted OpenStack, both the server running OpenStack and the OpenStack installation have to become trusted. The server running OpenStack can become trusted using NFVI trust as described in the previous Section. However, NFVI trust does not secure the OpenStack installation.

The two existing implementations build trust in operations running on top of OpenStack, but does not build trust in the OpenStack installation itself [34, 59]. There is a large body of research dedicated to OpenStack security [3, 7, 36, 45, 62, 63]; however, most of these focus on security issues other than integrity, such as access control and communication security [3, 45, 62, 63]. Two available research papers [7, 36] extend measurements into run time; however, they are mostly theoretical and do not measure the OpenStack installation.

3.5 VNF Trust

VNFs change state at many points during their life cycles. Therefore, their trust needs to be evaluated not only during VNF boot, but at every major state change. In this section, VNF trust is reviewed for adding images to the supply chain, starting a VNF and managing a VNF life cycle.

3.5.1 Supply Chain

Before VNFs, network functions were bound to hardware and securing an image could be done with physical security measures [13]. This is no longer the case since VNF images can be altered and run on any general purpose server.

In OpenStack, VNF images are added to an internal OpenStack database. This process is currently secured in a multitude of ways.

- Intel CIT hashes selected parts of the image and checks these parts during image boot [34].
- The master thesis trust implementation both signs and measures images [59]. The signature and measurement hashes are then verified before an image is started [59].
- The most common approach in research is to hash the entire image and check the hash against known values before an image is started [6, 13, 46].

3.5.2 Starting a VNF

In OpenStack, starting a VNF takes an image, selects a hypervisor and runs the image. Since VNF images are covered in the supply chain, the remaining element that needs trust is the hypervisor selection.

OpenStack select a hypervisor by going through a set of hypervisor filters that checks for sufficient computational resources, including the number of available CPUs and the amount of available RAM [21]. Among the hypervisors that satisfies the filter, the one with most available resources is chosen. The process of selecting a hypervisor is called workload placement.

Workload placement based on infrastructure trust has existed in earlier OpenStack versions [22]; however, it is now removed [16]. Both the current trusted NFV implementations adds trust to workload placement [34, 59]. In the existing implementations, trust in hypervisors is constructed and checked using measured boot and RA, as explained above under NFVI trust. Then a filter is added which removes all hypervisors that have failed the RA. Both of the current implementations rely on an outdated OpenStack, where the trusted filter still existed [22, 34, 59].

3.5.3 VNF Instance Life Cycle Operations

In additon to being trusted when started, VNFs need to maintain trust throughout their life cycles. OpenStack has a large set of VNF operations, which may change the VNF state. A full VNF state diagram is found in Appendix B Figure B1.

There are many theorized methods of securing a running VNF, including virtualized TPMs [2, 35], software TPMs [58] and IMA measurements within a running VNF [35]. Intel CIT is the only practical implementation that adds trust to VNF operations other than start VNF.

Intel CIT supports the following OpenStack VNF operations: launch from image, suspend, resume, hard reboot, shut off instance and start instance. All of these operations hash parts of the VNF image when the VNF instance is shut down or paused. These partial hashes are then verified when the VNF image is booted [34]. However, as neither suspend nor resume contains a boot action [21], it is unknown how, or if, the hashes are actually verified.

A larger set of VNF operations, such as migration [13, 26] and suspend [6, 46], are covered in patents; however, these are theoretical and directed at VMs in general.

3.6 Trusted NFVIaaS

While the rest of this Chapter has reviewed trust in NFV from an architectural view, this Section reviews trust in the trusted NFVIaaS use case. This use case was presented in Section 2.6 and aims to provide a cloud tenant with a trusted VNF instance. The remainder of this Section examines how trust can be added to the steps in the use case success scenario described in Section 2.6.1. For all the steps, a superset of the two existing implementations [34, 59] are used to provide trust.

Service provider adds trusted hardware elements to the cloud. Both of the existing implementations use measured boot and measures CRTM, SRTM and DRTM in a hardware element at boot time [34, 59]. With this, a service provider can guarantee the trust of an element at boot time.

Service provider installs a trusted OpenStack configuration. None of the existing implementations do run time attestation of software [34, 59]. Therefore, the OpenStack configuration can not be trusted, as the service provider can not validate its integrity.

Cloud tenant adds a trusted VNF image. The two existing implementations allows for different methods of trusting a VNF image. Intel CIT hashes parts of the image and verifies this hash during boot [34], while the master thesis implementation both signs and hashes the image and verifies these before boot [59]. Both of these methods allow a cloud tenant to add trust to their VNF image.

Cloud tenant deploys a VNF image on trusted hardware Both existing implementations provide what is known as trusted workload placement [34, 59]. This is done by modifying OpenStack filters to only run trusted VNF images on trusted hardware elements. With trusted workload placement, a cloud tenant can deploy VNF images on trusted hardware.

Cloud tenant performs trusted VNF operations. Only Intel CIT provides trusted VNF operations [34]. However, they only support a few operations and no operations where the VNF changes hypervisor [34]. Therefore, the cloud tenant can only perform a very limited set of trusted VNF operations.

Cloud tenant audits the trust decisions. The two existing implementations has some auditability. Intel CIT offers a trusted or not trusted decision for each element [34], while the master thesis implementation keeps an audit log over trust decisions [59]. Only relying on a trusted or not trusted decision allows for very limited auditing [34]. Although the alternate implementation offers an audit log, the implementation does not cover any VNF operations [59]. Without VNF operations, a cloud tenant can audit hardware element trust but not VNF instance trust.

3.7 Limitations in Existing Solutions

Although existing trusted NFV cloud solutions improve greatly on base OpenStack trust, they still have large limitations. Limitations for trust in an NFV cloud are listed below. Other limitations are out of the scope of this thesis and will not be considered.

- All existing implementations depend on Intel CIT for parts of their solution [34, 59].
- Boot time measurements are not extended into run time in any NFV cloud implementation [34, 59].
- OpenStack configuration is not measured in any OpenStack implementation or research [3, 7, 34, 36, 45, 59, 62, 63].
- Only a minimal set of OpenStack VNF operations are covered in existing implementations [34, 59]. Moreover, the available research is only theoretical and not adapted for VNFs on OpenStack [6, 13, 26, 46].
- Logging in current implementations is limited [34, 59], which makes auditing trust decisions difficult.

3.8 Summary

This Chapter has introduced existing trust implementations, presented trusted cloud challenges, reviewed previous trust research, applied existing trust solutions to NFVIaaS and uncovered limitations in existing solutions.

There is a large body of trust research, which enable a far higher level of trust than what was offered by a base OpenStack NFV platform. However, many trust issues remain unsolved and both the NFV architecture and the NFVIaaS use case have untrusted areas.

The next Chapter presents how this thesis plans to overcome the existing trust limitations and introduces the thesis extensions.

4 Proposed Extensions

This Chapter presents the extensions proposed by this thesis and explains how they solve or mitigate the current trusted NFV cloud limitations. To enable these extensions, an attestation server is added to the NFV architecture. This Chapter begins by introducing the attestation server, then it proceeds with the extensions in the order they were presented in Chapter 1.

In addition, each section will clarify how it solves or mitigates limitations introduced in Chapter 3.

4.1 Attestation Server

This thesis extends the NFV architecture by adding an attestation server. The attestation server attests hardware elements via remote attestation; attests VNFs and tracks VNF state via OpenStack VNFM; logs all trust events in the system and enables a user to audit all events.

The attestation server communicates with MANO elements from the NFV architecture shown in Chapter 2 Figure 2 and users. The functional blocks of the attestation server and its communication with the NFV architecture is shown in Figure 11. All blocks of the attestation server communicate with each other. The functional blocks not implemented in this thesis are used for ongoing research into more extensive attestation and root cause analysis [53]. Figure 12 shows a sample VNF instance in the full attestation server. Note that the graphical user interface was not developed for this thesis, but is a part of ongoing research [53].

The attestation server tracks the system trust status with a set of entries. These entries are different elements, quotes and policies. The attestation server elements are the OpenStack NFV entities introduces in Chapter 2 Section 2.5, which are hardware elements, VNF images and VNF instances. The attestation server determines the trust status of hardware elements using quotes and policies. Furthermore, it tracks the trust status and state of VNF images and VNF instances.

The attestation server also acts as a logging hub. All attestation actions in the cloud are logged in the attestation server. These logs are not actively used to determine trust; however, they allow any user to audit the trust decisions via command line tools.

This attestation server is different from Intel CIT and no parts of the solution depend on Intel CIT. This solves the limitation of all existing implementations having that as a single common element.

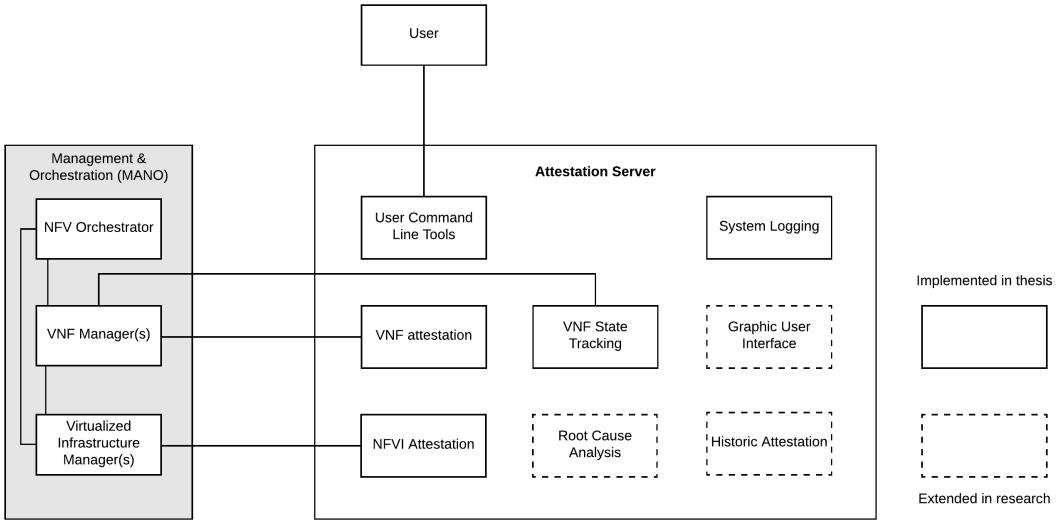

Figure 11: Attestation server functional blocks and communication with NFV MANO

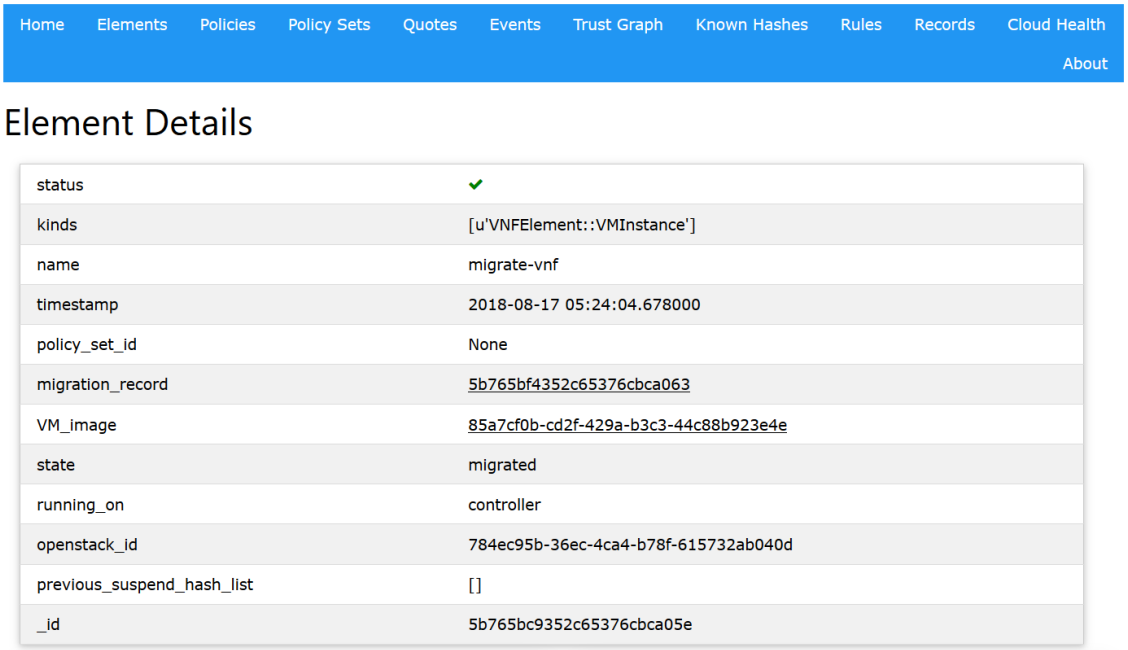

Figure 12: VNF instance entry in the attestation server

4.2 Extending Cloud Infrastructure measurements

This thesis extends measured boot and measures the OpenStack installation at run time. OpenStack operation is governed by a set of configuration files (the full set can be seen in Appendix B Section B.2) and only these needs to be measured to

ensure correct OpenStack operation. The OpenStack configuration files are tagged by SELinux and the tagged files are measured by Linux IMA. Linux IMA writes the hash of these files into a TPM PCR.

This measurement is used in two ways. The first way is when doing RA. When deciding the trust status of a hypervisor, also the run time measurements can be quoted, as it is written in TPM PCRs. The second way is to disable communication between OpenStack and the attestation server if OpenStack has changed. TPMs have the ability to seal data to a certain value in a PCR. This thesis seals the communication setup between OpenStack and the attestation server to a known good value of the IMA measured PCR.

This extension solves the limitations of boot time measurements not being extended into run time and the lack of OpenStack configuration measurements.

4.3 Extending VNF Life Cycle Trust

This thesis extends current solutions for measuring VNF life cycle operations. The VNF operations covered by this thesis are start, suspend, resume and migrate. This is still only a small subset of operations; however, most other operations are a variation of these. Some examples of similar operations are shelve and pause, which both use the same flow as suspend.

This trusted VNF extension does not only cover new VNF operations, but also adds to the supply chain of VNF images. In previous implementations, new images had to be added manually and were only used for the start VNF operation. This extension allows suspend and migrate to add new images to the supply chain, which can be used by OpenStack when resuming a suspended VNF or when migration has completed.

The base OpenStack start, suspend, resume and migrate can be seen in Chapter 2 Figures 4, 5, 6 and 7, while the extended OpenStack start, suspend, resume and migrate can be seen in Figures 13, 14, 15 and 16 respectively.

This extension mitigates the lack of OpenStack VNF operations covered in existing implementations and presents a flow that can easily be extended to a larger set of operations.

4.4 Extending the Auditability

This thesis extends the auditability of the cloud by logging all actions to the attestation server. These logs contains enough info so that a user could quote the different TPMs, hash the relevant VNF images and replay the operations taken to verify the

trust decisions themselves. Some of these audit capabilities existed in the previous master thesis solution [59], but in Intel CIT the user is left with little more than a trusted or not trusted decision [34].

When all actions are logged, a user can also find normal patterns for failure and attempt to track down the causes of these. One normal failure pattern in this cloud is due to synchronization failure in OpenStack compute nodes. This causes a compute node to not be available in workload scheduling, which would be difficult to discover without adequate logging.

This thesis only enables logging and it does not use the logs for any pattern recognition. The logging of all actions solves the last limitation from Chapter 3, which was the difficulty of auditing a trust decision.

4.5 Summary

This Chapter has described the extensions added by this thesis and clarified how they solve and mitigate current trusted NFV cloud limitations. In addition, it has presented the attestation server added to the NFV architecture. The next Chapter implements the extensions in a state-of-the-art trusted NFV cloud.

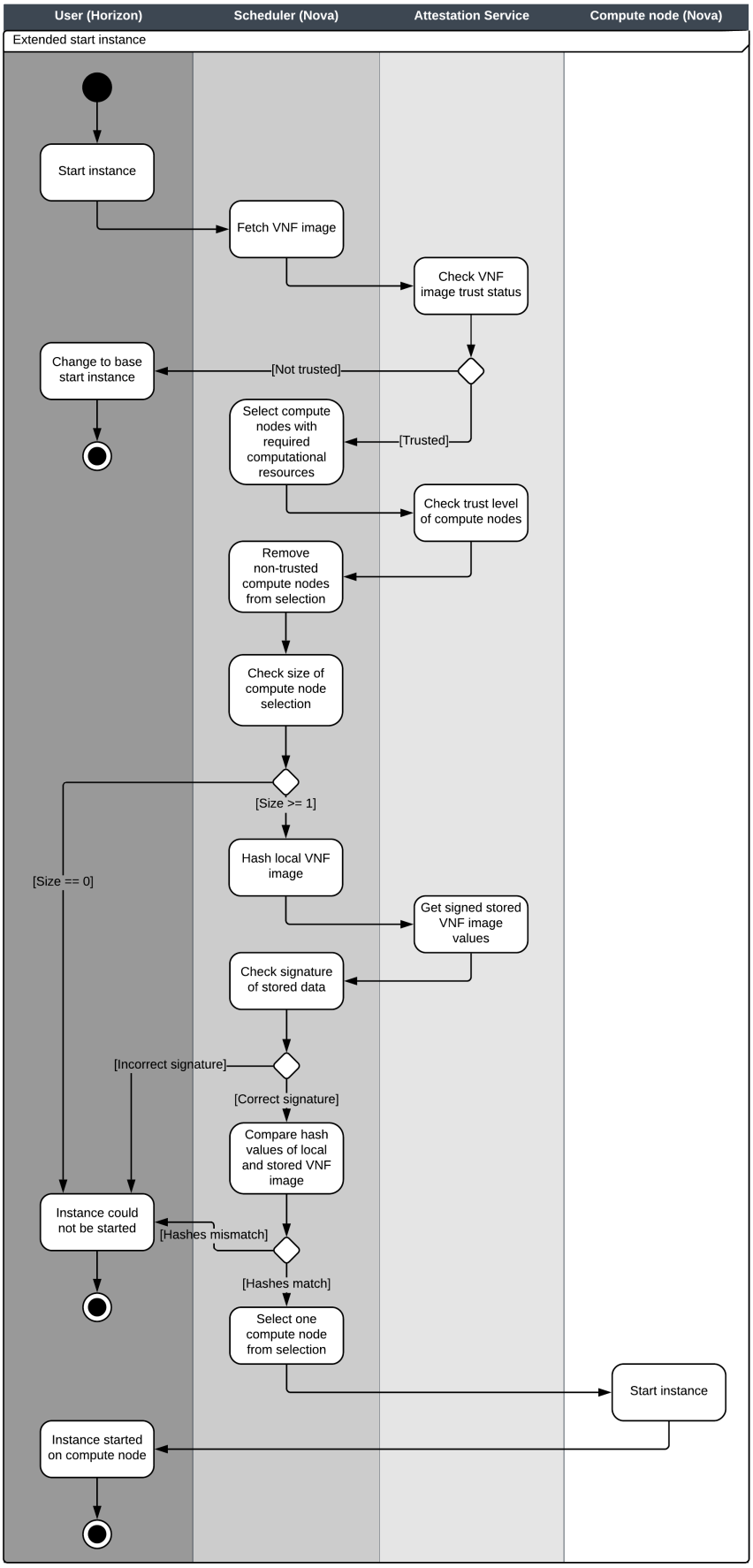

Figure 13: Extended OpenStack start VNF instance

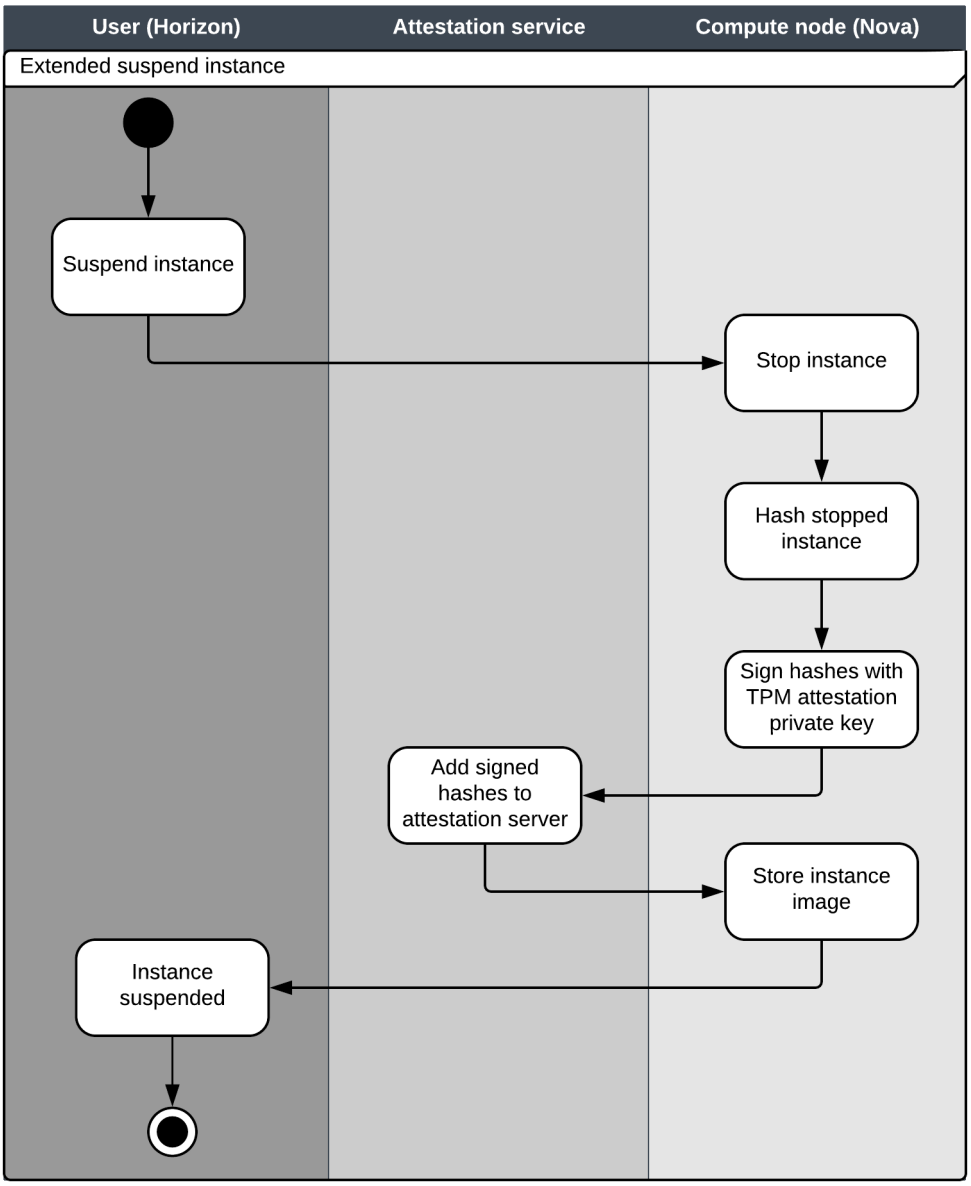

Figure 14: Extended OpenStack suspend VNF instance

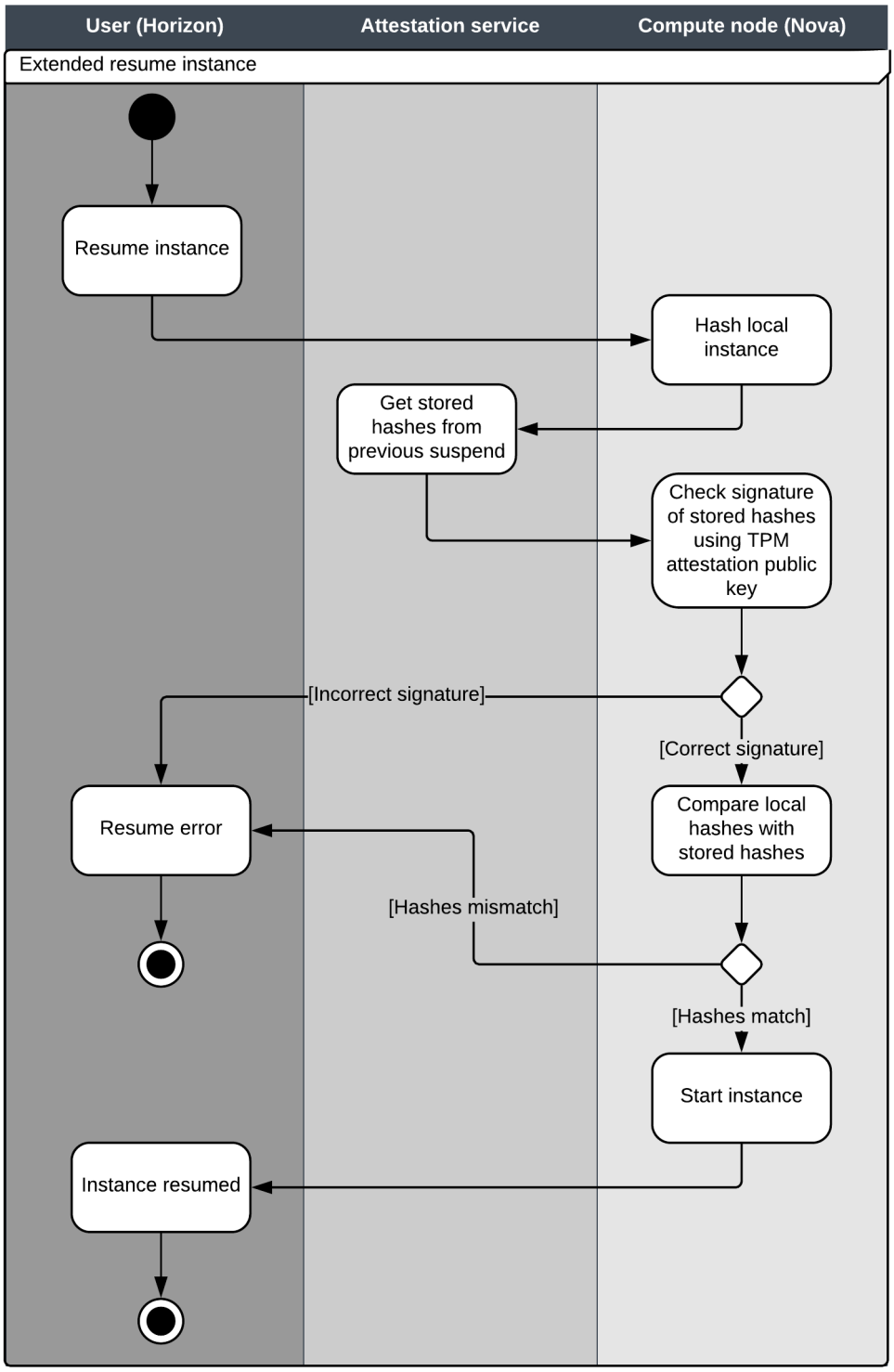

Figure 15: Extended OpenStack resume VNF instance

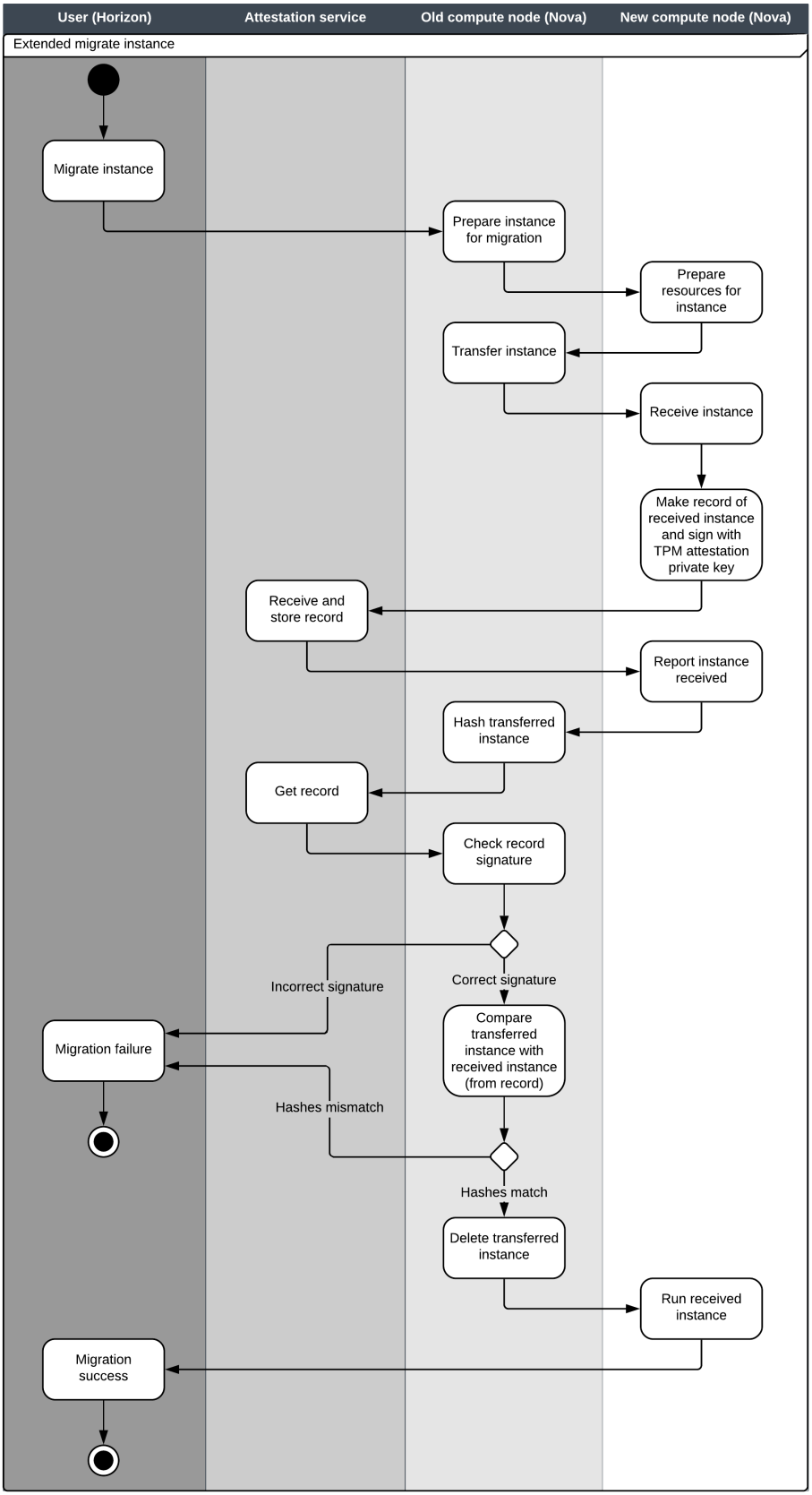

Figure 16: Extended OpenStack migrate VNF instance

5 Implementation

This Chapter presents a state-of-the-art trusted cloud implementation including the extensions proposed by this thesis. The implementation is a functional NFV cloud and includes a full OpenStack installation, running VNF instances and the trust elements introduced by previous implementations. The trust elements introduced by previous implementations are rebuilt in this thesis and no code is reused or copied.

This Chapter is organized by first describing the attestation server implementation before the trusted cloud is implemented from the bottom up architecture wise.

5.1 Attestation Server

This implementation extends the NFV architecture by adding an attestation server. The attestation server is used for storing, checking and auditing measurement values. The server communicates with the hypervisors managed by OpenStack and the OpenStack controller.

The attestation server stores entries for TPM elements, VNF image elements, VNF instance elements, policies, TPM quotes, events and records. These entries and their entry fields will be introduced in detail as they are used throughout this implementation. Example values for all entry fields can be found in Appendix D Section D.2.

The attestation server is developed in Python and communicates over HTTP requests via a representational state transfer (REST) application programming interface (API). A Python example of a HTTP request for getting OpenStack elements can be seen in Listing 1, while the full REST API is found in Appendix D Section D.1.

```
1 requests.get('http://' + ATTESTATION_SERVER_IP + ':' + str(
    ATTESTATION_SERVER_PORT) + '/elements/openstack/' +
    openstack_uuid)
```

Listing 1: Python HTTP request

The attestation server database is a MongoDB[18]; however, it is simply used for storing entries and could have been any database software. The main ID for database entries is their assigned MongoDB ID for policies, quotes, events and records and their OpenStack UUID for TPM elements, VNF image elements and VNF instance elements.

Database entries are signed when added. These signatures are done using

[18]https://www.mongodb.com/

OpenSSL[19]. Listing 2 shows an example OpenSSL signing command.

```
1 $ openssl dgst -sha256 -sign signing_key.key -out signed_example.txt
    example.txt
```

Listing 2: OpenSSL sign

For this implementation, the public keys of authorized signers are distributed manually.

5.2 Trusted NFVI

This implementation uses three NFVI elements, which all have a TPM and run OpenStack. All three servers function as OpenStack compute nodes, also known as hypervisors. Additionally, one server has the role of controller and therefore runs the MANO functionality of OpenStack. The OpenStack install was done according to the default install for the Pike release of OpenStack[20]. All the compute nodes are running the OpenStack Nova and Neutron services, while the controller node additionally runs the OpenStack Keystone, Glance and Horizon services. Appendix A shows the full server specification.

Each server is set up with the appropriate TPM drivers and tboot. SELinux has issues in Ubuntu; therefore, the Linux IMA and SELinux tests in Appendix C Section C.3 were performed on a different server running Fedora. However, SELinux is officially supported in Ubuntu and should function in future implementations [65]. The rest of this implementation uses IMA measurements on the AirFrame servers without accurate SELinux tagged OpenStack files.

Setup for tboot and Linux IMA can be found in Appendix C, while SELinux can be installed via your chosen package manager, such as Dandified YUM for Fedora Linux. With these programs, the servers have measurements for boot time CRTM, SRTM and DRTM and run time IMA.

The servers also have Intel's TPM software stack installed to simplify communication with the TPM. This software stack conforms to the TCG TPM 2 specification [30] and comprises TPM2-tss[21], TPM2-abrmd[22] and TPM2-tools[23]. The TPM2-tools are userland tools to simplify the communication with the TPM. In addition to the TPM software stack, each server runs a trust agent. The trust agent in this thesis is a simple Python wrapper that allows external elements to request TPM quotes from the TPM via the TPM software stack. Communicating to the TPM via userland

[19] https://www.openssl.org/
[20] https://docs.openstack.org/pike/install/
[21] http://github.com/intel/tpm2-tss
[22] https://github.com/intel/tpm2-abrmd
[23] https://github.com/intel/tpm2-tools

tools is not ideal and future implementations should communicate with the TPM directly. Figure 17 shows the implemented TPM communication and recommended future communication. The TPM 2 software stack is under active development and

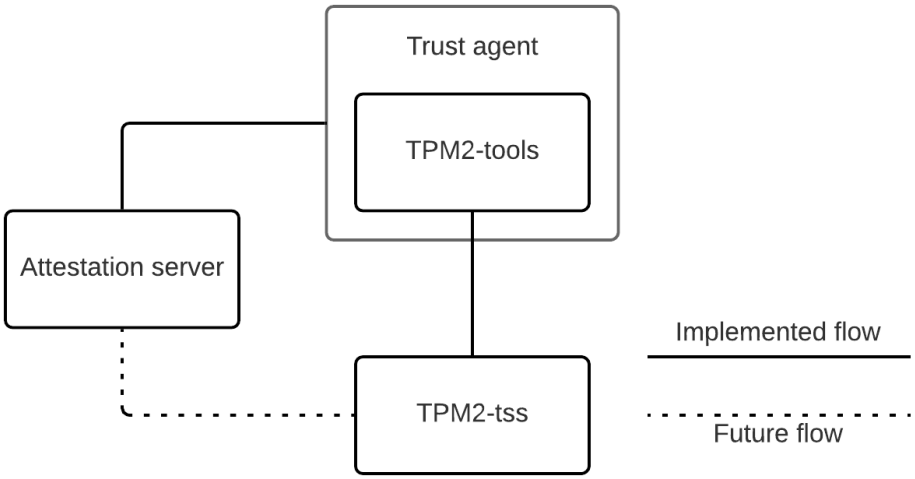

Figure 17: Attestation server TPM communication

this implementation uses the syntax from TPM2-tss version 1.4.0, TPM2-abrmd version 1.3.1 and TPM2-tools version 3.0.3. All TPM calls in this thesis are to this software stack, either directly or via the trust agent.

All of the servers are added to the attestation server as TPM elements. These attestation server entries include the identities of the servers, their configuration, their public TPM keys and their trust status. TPM element entry fields and their descriptions can be seen in Table 1. This thesis uses quotes and policies to determine the trust status of a TPM element.

A TPM quote is defined by the TCG [30] and contain the values of chosen TPM PCR registers and extra data, such as the TPM firmware version. This thesis only use the values of the quoted TPM PCRs. TPM quote entry fields and their descriptions are seen in Table 2. Quotes are taken using the `tpm2_quote` command from Intel's TPM2 software stack. An example quote command for SHA-256 PCR registers 0–7 is shown in Listing 3.

```
1 $ tpm2_quote -k 0x81010003 -L sha256:0,1,2,3,4,5,6,7 -m message.
    output -s signature.output -G sha256
```

Listing 3: TPM quote command

A policy gives an expected value for a specific measurement. In this thesis, all policies are for TPM elements and have a set of PCRs and their expected value. To

Field name	Description
_id	MongoDB ID of the TPM element.
ak	The public part of the TPM element's AK.
ek	The public part of the TPM element's EK.
ip	IP of the TPM element.
kind	What kind of element it is.
name	Name of the TPM element.
openstack_id	OpenStack UUID of the TPM element.
status	Trust status of the TPM element.
timestamp	Time when this entry was made to the database.
uname	Info about the TPM element platform.
policy_list	List of policies for this element.
signature	Signature of a system administrator.

Table 1: TPM element fields

Field name	Description
_id	MongoDB ID of the TPM quote.
element_id	The ID of the TPM element on which the quote was taken.
quote	The actual TPM quote.

Table 2: TPM quote fields

find expected values, this thesis uses reference devices. A reference device is made by configuring a server to a known good configuration and quoting sets of PCRs. The quoted value is then stored as the expected value for the quoted PCRs. Policy entry fields and their descriptions are seen in Table 3.

Field name	Description
_id	MongoDB ID of the policy.
name	Name of the policy.
timestamp	Time when this entry was made to the database.
pcrs	Which PCRs this policy is for.
expected_value	The expected value of the PCRs.
signature	Signature of a system administrator.

Table 3: Policy fields

Trust is determined in a TPM element by checking its policies. To check the policies, a TPM element is quoted for the PCRs required by its policies. These quoted values are then compared with the expected values in the policies. If these are equal, then the TPM element is set to trusted. The full process determining TPM element trust is described below.

- Get element contact info from a TPM element entry.
- See which policies are in the elements policy list.
- For each policy, ask the element to upload a quote of the required PCRs.
- Check the added quote entry signatures with the public TPM AK of the element. An example command for checking a TPM signature is found in Listing 4.
- For each policy, check the quoted value against the expected value.
- If all policy checks are successful, then the TPM element is marked as trusted

```
1 $ tpm2_verifysignature -k 0x81010003 -g sha256 -m verify_input -s
    signature.sig -t validation.ticket
```

Listing 4: TPM quote command

5.3 Trusted MANO

In this thesis, OpenStack implements the MANO functionality. Therefore, MANO trust is built by enabling trust in the OpenStack controller installation. OpenStack

controller functionality is determined by a number of configuration files so we can create a trusted MANO by trusting the content of these configuration files.

These configuration files can be measured by Linux IMA. Since IMA itself lacks an accurate way of tagging files, this implementation uses SELinux tags to mark the configuration files. IMA can then define a policy where IMA measures all files with a certain SELinux tag. Appendix C Section C.3 shows the process of tagging a single OpenStack configuration file for IMA measurement, while Appendix B Section B.2 shows the full list of OpenStack configuration files.

The process described in Appendix C Section C.3 measures OpenStack configuration files and saves the measurements in TPM PCR 14. Other PCR numbers could also have been chosen. To make MANO trusted, attestation server communication is made dependant on PCR 14 remaining at a known good value. This is done by protecting the communication configuration in TPM NVRAM. The configuration for this thesis is only the IP of the attestation server; however, for a future implementation this could have been a shared secret key. The steps to add configuration data to TPM NVRAM is shown below, while the code to perform the steps is shown in Listing 5.

- Write PCR 14 to an output file.
- Create a policy for the output file.
- Find space for the policy by checking the TPM NVRAM. The code in Listing 5 checks available memory and releases the `0x1800007` memory location. This is done under the owner hierarchy, stored in memory location `0x40000001`.
- Define the policy.
- Choose the file to protect with the policy.
- The trust configuration is now stored in TPM NVRAM and can only be read if PCR 14 has not changed.
- The file can later be read with `tpm2_nvread`.

When these steps are taken, OpenStack configuration files are measured and the communication configuration between OpenStack and the attestation server is sealed against this measurement. This entails that OpenStack can only communicate with the attestation server if its configurations files have not been modified. The attestation server is needed for all VNF operations and losing communication will prevent OpenStack from starting or changing VNFs.

```
$ tpm2_pcrlist -L sha1:14 -o output_file
$ tpm2_createpolicy -P -L sha1:14 -F output_file -f policy_name
$ tpm2_nvlist
$ tpm2_nvrelease -x 0x1800007 -a 0x40000001
$ tpm2_nvdefine -x 0x1800007 -a 0x40000001 -s 700 -L policy_name -t "
   policyread||policywrite"
$ tpm2_nvwrite -x 0x1800007 -a 0x1800007 -L sha1:14 -F output_file
   trust_configuration
$ tpm2_nvread -x 0x1800007 -a 0x1800007 -L sha1:14
```

Listing 5: TPM sealing commands

5.4 Trusted Supply Chain

To be able to run trusted workload in OpenStack, images are added to the attestation server. The image entry fields can be seen in Table 4. These images have an ID and

Field name	Description
_id	Database ID of the VNF image element.
hash_list	A signed list of expected hashes for the VNF image element.
kinds	What kind of element it is.
name	Name of the VNF image element.
openstack_id	OpenStack UUID of the VNF image element.
status	Trust status of the VNF image element.
timestamp	Time when this entry was made to the database.
signature	Signature of a system administrator.

Table 4: VNF image element fields

a set of hash methods and their expected values. At least one hashing method need to be available and this implementation supports the hash algorithms MD5, SHA-1 and SHA-256.

When starting a VNF instance, OpenStack checks the metadata of the chosen VNF image. This thesis adds metadata to OpenStack VNF images with the following steps.

- Add a trusted field to `/usr/lib/python2.7/dist-packages/nova/objects/fields.py`.

  —

```
class Trusted(BaseNovaEnum):
    TRUE = 'True'
    FALSE = 'False'
    ALL = (TRUE, FALSE)

class TrustedField(BaseEnumField):
    AUTO_TYPE = Trusted()
```

- Add a metadata trusted property to `/usr/lib/python2.7/dist-packages/nova/objects/image_meta.py`.

  —

```
class ImageMetaProps(base.NovaObject):
...
```

```
dictionary fields = {
'trusted': fields.TrustedField(),
...
}
```

- Set the metadata of an image with `$ openstack image set --property trusted='True' image_name`

This metadata determines if the VNF image requires to be launched on a trusted OpenStack hypervisor. Any image marked as not trusted will ignore added functionality and will not run on trusted hypervisors.

Adding metadata and measurements to VNF images greatly improves the supply chain trust. However, important supply chain elements outside of OpenStack, such as VNF descriptors, are not covered by this implementation.

5.5 OpenStack Trust Filter

OpenStack filters are used to schedule VNFs starting in OpenStack. Filters are stored in `/usr/lib/python2.7/dist-packages/nova/scheduler/filters/` and this thesis adds a filter called `trusted_filter`. This filter is enabled in OpenStack Pike by adding the following code to `/etc/nova/nova.conf`.

```
[scheduler]
...
driver = filter_scheduler

[filter_scheduler]
...
available_filters=nova.scheduler.filters.all_filters
available_filters=nova.scheduler.filters.trusted_filter
enabled_filters=ComputeFilter, TrustedFilter
```

The added filter is run as part of the OpenStack start VNF instance operation. Base OpenStack start instance activity diagram can be seen in Chapter 2 Figure 4 and the extended process with an added trust filter can be seen in Chapter 4 Figure 13.

To run a filter, it needs to implement the `host_passes` method. An absolute minimum filter showing the syntax for getting host and image info from a filter and returning the trusted property of an image is shown in Listing 6 and the code for getting an image out of the Glance repository is shown in Listing 7.

The remaining code in the Python filter is for creating an event (example events shown in Appendix D Section D.3), verifying the trust status of hypervisors through

the attestation server REST API (shown in Appendix D Section D.1) and doing comparisons of measured values and expected values.

```
class TrustedFilter(filters.BaseHostFilter):
       def host_passes(self, host_state, spec_obj):
              hypervisor_name = host_state.host
              hypervisor_openstack_id = host_state.uuid
              vnf_image_openstack_id = spec_obj.image.id
              vnf_instance_openstack_id = spec_obj.instance_uuid
              return spec_obj.image.properties.trusted
```

Listing 6: Example of OpenStack filter syntax

```
auth = v3.Password(
       auth_url='OPENSTACK_CONTROLLER_IP:5000/v3',
       username=GLANCE_USER,
       password=GLANCE_PASSWORD,
       project_name="admin",
       project_domain_id="default",
       user_domain_id="default"
       )
sess = session.Session(auth=auth)
glance = Client('2', session=sess)
image = glance.images.data(vnf_image_openstack_id)
```

Listing 7: Getting a Glance image from a Python filter

5.6 Trusted OpenStack Operations

To enable trusted OpenStack operations, this thesis intercepts the normal RabbitMQ flow shown in Chapter 2 Figure 3. RabbitMQ flow is changed using RabbitMQ management command line tool[24]. This interception adds a trusted step before any message is able to reach a consumer.

To intercept RabbitMQ messages, all the three compute nodes are thrown out of the `Nova` exchange and added to a new exchange called `Trusted`. New consumers are added on each node and these consumers perform the trust checks. The new consumers consume messages from the `Nova` exchange and publish messages that have passed the trust check on the `Trusted` exchange. Messages that do not pass the

[24]https://www.rabbitmq.com/management-cli.html

trust checks do not get passed to the compute nodes, instead the added consumers create an error event.

RabbitMQ syntax uses nested Python dictionaries which vary depending on the message. The syntax for getting VNF instance info out of a RabbitMQ message for a VNF operation is shown in Listing 8.

```
1 def callback(ch, method, properties, body):
2       message_dict = json.loads(body_dict['oslo.message'])
3       object_dict = message_dict['args']['instance']['nova_object.
           data']
4       openstack_operation_name = message_dict['method']
5       openstack_instance_uuid = object_dict['uuid']
6       openstack_instance_state = object_dict['vm_state']
```

Listing 8: Example of RabbitMQ message syntax

RabbitMQ does not only send messages for VNF operations, but also events named `external_instance_event`. These events signal completed VNF operations and state changes in VNF instances.

The steps added to VNF start, suspension, resume and migrate are shown in the extended activity diagrams in Chapter 4 Figures 13, 14, 15 and 16 respectively. The Python code for trust checks in extended VNF operations is similar to the code for performing trust checks in OpenStack filters; therefore, it is not documented in greater detail.

In addition to doing trust checks, the new trusted RabbitMQ consumers add entries to the attestation server. These entries are VNF instance entries, shown in Table 5, and migration records entries, shown in Table 6. The VNF instance entries tracks the status and state of VNF instances at all times and gets updated according to data found in RabbitMQ messages. Furthermore, the VNF instance entries are used to keep the hash values of a VNF image upon suspension and to link VNF instances to migration records. The migration record entry is used to verify that the migrated VNF image is the correct one and that the receiver is the correct hypervisor. Both of these added entries are signed with the hypervisors TPM. The TPM commands for hashing and signing are shown in Listing 9.

```
1 $ tpm2_hash -H e -g sha256 -o hash_output.hash -t validation.ticket
     hash_input
2 $ tpm2_sign -k 0x81010003 -m sign_input -s signature.sig -g sha256 -t
      validation.ticket
```

Listing 9: TPM hash and sign

Field name	Description
_id	Database ID of the VNF instance element.
previous_suspend_hash_list	A list of expected hashes renewed every VNF instance suspension.
kind	What kind of entry it is.
name	Name of the VNF instance element.
openstack_id	OpenStack UUID of the VNF instance element.
status	Trust status of the VNF instance element.
timestamp	Time when this entry was made to the database.
original_image_openstack_id	OpenStack UUID of the VNF image element this VNF instance element is based upon.
state	The current state of the VNF instance.
migration_record	The record of any migration actions taken by the VNF instance.
running_on	The element on which the VNF instance runs.
signature	Signature of the TPM AK from the element that added the entry.

Table 5: VNF instance element fields

_id	Database ID of the migration record.
completed	If the instance has started on the new hypervisor.
from_host	The hypervisor the instance has migrated from.
to_host	The hypervisor the instance has migrated to.
from_instance	The instance entry pre-migration.
to_instance	The instance entry post-migration.
kind	What kind of entry it is.
openstack_id	OpenStack UUID of the VNF instance element.
status	Trust status of the migration.
timestamp	Time when this entry was made to the database.
transfer_hash_list	A list received VNF instance hashes.
signature	Signature of the TPM AK from the element that added the entry.

Table 6: Migration record fields

5.7 User Auditing

All the actions described in this chapter creates events. These events are sent to the attestation server through its REST API and are meant for enabling user auditing of the system. By looking at the events, a user can verify the information used for a trust decision and do not have to trust the central attestation server. There are many different events, as they document all actions, and some example events can be found in Appendix D Section D.3. This thesis does not automate any event reasoning.

5.8 Summary

This Chapter has shown how to implement a state-of-the-art trusted NFV cloud. The implementation included elements from base OpenStack, from previous trusted cloud solutions and all the extensions proposed in Chapter 4.

This extended trusted cloud implementation added an attestation server to the NFV architecture. The attestation server was used for storing and verifying measurements at critical points during the cloud operation. These measurements checks were implemented for hardware elements, VNF images and VNF instances. Hardware elements were checked during boot time and run time, VNF images were checked during initialization and deployment while VNF instances were checked during a set of VNF operations. All the trust information gained by the measurement checks were distributed and made auditable through the attestation server.

The next Chapter evaluates the trusted cloud extensions and this implementation.

6 Evaluation

This Chapter evaluates the trusted NFV implementation and trust extensions added by this thesis. The evaluation criteria was given in Chapter 1, were this thesis defined trust as the confidence in the integrity of hardware and software throughout their life cycles and set the aim to fulfill the NFV specified goal to integrity protect hardware and software [11].

Part of this evaluation has been done in previous chapters. Chapter 3 reviewed current solutions and discovered several limitations. These limitations were solved or mitigated through extensions introduced in Chapter 4 and implemented in Chapter 5. With these implemented extensions, this thesis has created a stronger confidence in the integrity of hardware and software in an NFV cloud.

The remainder of this Chapter finishes this evaluation, reviews trusted NFVI-aaS with the added trust extensions and measures the performance of the thesis implementation.

6.1 Improvements over Existing Solutions

Existing trusted cloud solutions have several limitations that were described in Section 3.7. All of these have been solved or mitigated by this thesis.

All existing implementations depend on Intel CIT for parts of their solution. The thesis does not depend on Intel CIT for any parts of its solution.

Boot time measurements are not extended into run time in any NFV cloud implementation. This thesis extends boot time measurements into run time with Linux IMA and SELinux.

OpenStack configuration is not measured in any OpenStack implementation or research. This thesis measures the OpenStack configuration with run time measurements.

Only a minimal set of OpenStack VNF operations are covered in existing implementations. This thesis extends the set of trusted OpenStack VNF operations and introduces a new method to support an even larger set of operations. However, this thesis does not cover all OpenStack operations.

Logging in current implementations is limited, which makes auditing trust decisions difficult. This thesis adds extensive logging to all trust operations and allows for extensive auditing of trust decisions via an attestation server.

6.2 Trusted NFVIaaS

While earlier thesis Sections have discovered and mitigated NFV trust limitations from an architectural view, this Section performs the final evaluation of the trusted NFVIaaS use case. The use case was presented in Section 2.6 and extended in Section 3.6. The steps below describe the NFVIaaS success scenario steps from Section 2.6.1 and compares them to previous state-of-the-art.

Service provider adds trusted hardware elements to the cloud. The implementation in this thesis measures CRTM, SRTM and DRTM in a hardware element. This was also done in earlier implementations [34, 59]. With these measurements, a service provider can add trusted cloud hardware elements.

Service provider installs a trusted OpenStack configuration. This thesis adds run time measurements that measures the integrity of a OpenStack installation and shuts down VNF operations if any OpenStack configuration file changes. No run time measurements were done in existing implementations [34, 59].

Therefore, a service provider now has the new option of installing trusted OpenStack software.

Cloud tenant adds a trusted VNF image. This thesis adds VNF images using both measurements and signatures. In addition, new VNF images are created as a result of suspend and migrate. Earlier implementations had either just measurements [34] or a combination of measurements and signatures [59]. None of them added new VNF images due to VNF operations.

A cloud tenant now has the same amount of trust in its own added images as current state-of-the-art. This thesis adds trusted VNF images that are automatically created and added to the cloud as needed. This enables more VNF operations, but does not affect cloud tenant added VNF images.

Cloud tenant deploys a VNF image on trusted hardware. This thesis adds trusted workload placement that depend not only on hardware element boot time measurements, but also run time measurements. The run time measurements extend the level of trust offered by existing solutions [34, 59].

As with previous solutions, a cloud tenant can deploy VNF images on trusted hardware. However, the hardware trust level has been increased.

Cloud tenant performs trusted VNF operations. This thesis added more VNF operations to the trusted NFV cloud, including changing hypervisor through migration. Existing implementations covered either no VNF operations [59] or a small subset of operations on the same hypervisor [34].

With this thesis, a cloud tenant can perform more trusted VNF operations than before.

Cloud tenant audits the trust decisions. This thesis added auditablity to all trusted actions, both by the trust solution and by the VNF instances. This improved existing implementations who either provided no audit log [34] or only an audit log of hardware actions [59].

With the implementation in this thesis, a cloud tenant can audit all trust decisions taken in the NFV cloud.

6.3 Performance

To measure the performance of the implementation, all covered VNF operations have been timed. This Section first measures the performance, then evaluates the result.

6.3.1 Performance Measurements Environment

The performance test were taken in the running OpenStack implementation. The OpenStack servers were Nokia AirFrame machines and their full specification can be found in Appendix A. The images used for time testing were three different Linux distributions and their specification is seen in Table 7. All tests were run on a local network, so any network latency would be minimal.

Image name	Linux distribution	Size
Small	Cirros	12.65 MB
Medium	Ubuntu	276.56 MB
Large	CentOS	1.27 GB

Table 7: Images used for time testing

6.3.2 Performance Measurements

Measurements were taken not only for the extended VNF operations proposed by this thesis, but also for the individual steps in these operations.

The time usage of base OpenStack operations and extended operations varied; therefore, the stated time is the average of 10 runs. The extended operations were done using only a SHA-256 hash and not the complete set of MD5, SHA-1 and SHA-256. All the operations were performed on the same servers, and the servers were not running any workload beyond OpenStack. Start, suspend and resume were done on AirFrame 2 while migration was done from AirFrame 2 to AirFrame 3.

Table 8 compares the total time and time difference of OpenStack operations both with and without extensions. Table 9 and Table 10 shows the time for the separate operations.

Operation name	Base average time	Extended average time	Percentage added
Start small image	11.21s	12.72s	13,47 %
Start medium image	18.31s	21.98s	20,04 %
Start large image	24.56s	33.23s	35,30 %
Suspend small image	5.34s	8.26s	54,68 %
Suspend medium image	9.11s	13.64s	49,73 %
Suspend large image	9.86s	19.36s	96,35 %
Resume small image	5.28s	7.07s	33,90 %
Resume medium image	9.12s	12.52s	37,28 %
Resume large image	9.76s	19.24s	97,13 %
Migration small image	20.54s	25.27s	23,03 %
Migration medium image	50.37s	56.34s	11,85 %
Migration large image	62.82s	74.16s	18,05 %

Table 8: Time difference of base OpenStack operations and extended OpenStack operations

Operation	Time small image	Time medium image	Time large image
MD5 hash	0.06s	0.70s	2.52s
SHA-1 hash	0.07s	0.79s	3.02s
SHA-256 hash	0.13s	1.75s	7.14s

Table 9: Time for image dependent operations

6.3.3 Performance Evaluation

The implemented extensions have a large effect on performance and the percentage of added time ranges from 11.85 % to 97.13 %. Most of the added time comes from image hashing and TPM operations. Therefore, the time added is related mostly to image size and not to the operation performed.

Operation	Time
TPM SHA-256 hash	1.02s
TPM SHA-256 quote	0.68s
TPM SHA-256 sign	1.79s
TPM SHA-256 verify	1.69s
OpenSSL SHA-256 sign	0.03s
OpenSSL SHA-256 verify	0.01s

Table 10: Time for image independent operations

The added time can be minimised by signing with OpenSSL or by using a quicker hashing algorithm. However, this will lead to a lower level of trust as the TPM is used for host verification and quicker hashing algorithms are easier to break.

Although the implemented extension have a large effect on performance, the most impacted VNF operations do not affect the telecommunications consumer directly. Both start, suspend and resume instance will mostly affect power usage for the cloud provider. The only measured VNF operation directly affecting the consumer is migration. Migration has the smallest percentage of added time and it might also be the operation benefiting the most from added trust due to changing hypervisor.

The level of trust needed should be balanced against the need for highest possible performance and not all VNFs benefit from running as trusted. It is also possible to only add trust to some operations.

6.4 Summary

This Chapter has evaluated the added extensions and implementation of this thesis. All the trust goals were met and all the NFVIaaS success scenario steps were fulfilled. Although the trust goals were met, the performance cost for VNF operations were high. However, the performance impact affect mostly the cloud provider. Nevertheless, the trust and performance trade-off should be considered carefully for each VNF. The next Chapter concludes this thesis and presents future work.

7 Conclusion and Future Work

This Chapter concludes the thesis and proposes future work. The conclusion summarizes the thesis aim, method and results, while future work outlines how to improve the solution in this thesis and extend it further.

7.1 Conclusion

This thesis has investigated how to add trust to a NFV cloud. We have looked into the NFV architecture and how it is most commonly implemented in OpenStack. In NFV on OpenStack there were three high level architectural concepts that needed to become trusted. These were the NFVI, MANO and VNF concepts. In addition we have reviewed trust in the NFVIaaS use case. This use case required trust in the OpenStack NFV entities, which were hardware elements, VNF images and VNF instances.

The NFVI comprises the hypervisors running the NFV cloud. We reviewed how hypervisor trust was added by earlier implementations and existing research. The most popular approach was using boot time measurements stored in a TPM. This thesis added another layer to this by introducing run time measurements. Run time measurements were done with Linux IMA.

MANO is the management layer of NFV. In OPNFV on OpenStack, MANO refers to the OpenStack controller. The OpenStack controller actions are governed by configuration files which can not be measured at boot time. None of the existing implementations has extended boot time measurements into run time. Even though some existing research utilizes run time measurement of files, no research measures the OpenStack installation itself. This thesis added trust to MANO by tagging files with SELinux and measuring them with Linux IMA. These measurements were used to seal important configuration in the TPM so that all VNF operations were shut down if the OpenStack configuration changed.

NFV virtualizes the network functions of telecommunications and allows them to run as VNFs, which are the virtual workload running on the NFVI. VNF trust was implemented in previous implementations only for a very limited number of operations and operations such as VNF migrate were not covered. This thesis introduced a method of adding trust to VNF operations by intercepting the internal communication of OpenStack. This new method was applied to a larger set of VNF operations than earlier covered and provided an alternative to modifying OpenStack source code. This method could easily be extended to cover most OpenStack operations.

In addition to adding trust to NFVI, MANO and VNFs, this thesis added an attestation server. This server tracked all the cloud elements and created events of

trust decisions made by the server. These events were made available to all users. In previous implementations, users had limited options to review trust decisions. With the attestation server events, users can redo the measurements themselves and do not have to blindly trust the attestation server.

This thesis aimed to enable a higher level of integrity protection of the hardware and software in a NFV cloud. This aim was evaluated by measuring the level of trust in the NFV architecture and in the NFVIaaS use case. For both of these, the trust aim was reached. However, the performance costs for VNF operations were quite high. Therefore, this thesis suggests that the proposed trust extensions should be used in situations were increased trust is more important than maximized performance or when VNF operations are rarely performed.

7.2 Future Work

There are large possibilities for future work in creating trusted NFV clouds, as NFV itself is quite recent. This Section proposes future work within the thesis scope of integrity measurements. Other important security features are out of scope, including secure communication and access control.

Infrastructure trust is in this thesis only based on TPM 2.0. Future work should expand on infrastructure trust and enable other security architectures, such as other hardware or software security modules. Infrastructure trust can also be expanded into virtual machines, through technologies such as virtual TPMs or software TPMs.

This thesis made the simplification that VNFs and VMs had a one-to-one relation. Future work should also cover the cases where a VNF is deployed over multiple VMs and also include extra VNF configuration data.

OpenStack is not the only NFV platform and trust in other platforms should also be considered. Especially Docker is relevant for telecommunications and trust concepts explored in this thesis would also be important for Docker.

A fully functional NFV cloud would have smaller physical edge elements in addition to the NFVI infrastructure servers. Future work should expand trust to these more resource constrained elements.

Previous trusted NFV implementations and this thesis only consider trust as a binary trusted or not trusted decision. Future work could remove this binary limitation and develop scalable trust.

Finally, the events created by this thesis implementation can be utilized to reason over the system, for example to find the root cause of failures. This could ease in error correction and help system administrators find the errors in their systems.

Bibliography

[1] Tigist Abera, N Asokan, Lucas Davi, Farinaz Koushanfar, Andrew Paverd, Ahmad-Reza Sadeghi, and Gene Tsudik. Things, trouble, trust: on building trust in IoT systems. In *Proceedings of the 53rd Annual Design Automation Conference*, page 121. ACM, 2016.

[2] Mohammed Achemlal, Said Gharout, and Chrystel Gaber. Trusted platform module as an enabler for security in cloud computing. In *Network and Information Systems Security (SAR-SSI), 2011 Conference on*, pages 1–6. IEEE, 2011.

[3] Hala Albaroodi, Selvakumar Manickam, and Parminder Singh. Critical review of OpenStack security: Issues and weaknesses. *Journal of Computer Science*, 10(1):23–33, 2014.

[4] Mohamed Almorsy, John Grundy, and Ingo Müller. An analysis of the cloud computing security problem. *arXiv preprint arXiv:1609.01107*, 2016.

[5] Will Arthur and David Challener. *A Practical Guide to TPM 2.0: Using the Trusted Platform Module in the New Age of Security*. Apress, 2015.

[6] Nir Barak, Amir Jerbi, Eitan Hadar, and Michael Kletskin. System and method for enforcement of security controls on virtual machines throughout life cycle state changes, July 12 2016. US Patent 9,389,898.

[7] Stefan Berger, Kenneth Goldman, Dimitrios Pendarakis, David Safford, Enriquillo Valdez, and Mimi Zohar. Scalable attestation: A step toward secure and trusted clouds. In *Cloud Engineering (IC2E), 2015 IEEE International Conference on*, pages 185–194. IEEE, 2015.

[8] ETSI. Network Functions Virtualisation (NFV); Architectural Framework. 2013. http://www.etsi.org/deliver/etsi_gs/nfv/001_099/002/01.01.01_60/gs_nfv002v010101p.pdf.

[9] ETSI. Nfv, network functions virtualisation and use cases, gs nfv 001 v1. 1.1. 2013.

[10] ETSI. Network Functions Virtualisation – White Paper #3. 2014. https://portal.etsi.org/Portals/0/TBpages/NFV/Docs/NFV_White_Paper3.pdf.

[11] ETSI. Network Functions Virtualisation (NFV); NFV Security; Security and Trust Guidance. 2014. http://www.etsi.org/deliver/etsi_gs/NFV-SEC/001_099/003/01.01.01_60/gs_NFV-SEC003v010101p.pdf.

[12] Anders Fongen and Federico Mancini. Integrity attestation in military IoT. In *Internet of Things (WF-IoT), 2015 IEEE 2nd World Forum on*, pages 484–489. IEEE, 2015.

[13] Ronald James Forrester, William Wyatt Starnes, and Frank A Tycksen Jr. Method and apparatus for lifecycle integrity verification of virtual machines, September 20 2016. US Patent 9,450,966.

[14] Linux Foundation. OPNFV & OpenStack. https://www.opnfv.org/community/upstream-projects/openstack Accessed 28.05.2018.

[15] OpenStack Foundation. Glance documentation. https://docs.openstack.org/glance/pike/index.html Accessed 15.08.2018.

[16] OpenStack Foundation. Hardening Compute deployments. https://docs.openstack.org/security-guide/compute/hardening-deployments.html Accessed 28.05.2018.

[17] OpenStack Foundation. Hypervisor selection. https://releases.openstack.org/ Accessed: 04.06.2018.

[18] OpenStack Foundation. Image Signature Verification. https://docs.openstack.org/glance/pike/user/signature.html Accessed 15.08.2018.

[19] OpenStack Foundation. Integrity life-cycle. https://docs.openstack.org/security-guide/management/integrity-life-cycle.html Accessed: 15.08.2018.

[20] OpenStack Foundation. Integrity life-cycle. https://docs.openstack.org/security-guide/management/integrity-life-cycle.html Accessed 28.05.2018.

[21] OpenStack Foundation. OpenStack Documentation. https://docs.openstack.org/pike/ Accessed: 27.06.2018.

[22] OpenStack Foundation. Security hardening. https://docs.openstack.org/mitaka/admin-guide/compute-security.html Accessed 28.05.2018.

[23] OpenStack Foundation. Accelerating NFV Delivery with OpenStack. 2016. https://www.openstack.org/telecoms-and-nfv/.

[24] The Linux Foundation. OPNFV Security Guide. 2016. http://artifacts.opnfv.org/opnfvdocs/docs/opnfvsecguide/opnfvsecguide.pdf.

[25] William Futral and James Greene. *Intel Trusted Execution Technology for Server Platforms: A Guide to More Secure Datacenters*. Apress, 2013.

[26] Christian Gehrmann, Mats Näslund, and Makan Pourzandi. Secure cloud-based virtual machine migration, April 18 2013. US Patent App. 13/275,722.

[27] Kenneth Goldman, Ronald Perez, and Reiner Sailer. Linking remote attestation to secure tunnel endpoints. In *Proceedings of the first ACM workshop on Scalable trusted computing*, pages 21–24. ACM, 2006.

[28] J Greene. Intel Trusted Execution Technology, white paper. *https://www.intel.com/content/dam/www/public/us/en/documents/white-papers/trusted-execution-technology-security-paper.pdf*, 2012.

[29] Trusted Computing Group. The Mobile Platform Work Group. https://trustedcomputinggroup.org/work-groups/ Accessed 28.05.2018.

[30] Trusted Computing Group. TPM library specification. 2014. https://trustedcomputinggroup.org/tpm-library-specification/.

[31] Thomas Haeberlen and Lionel Dupré. The Rise of Cloud Computing in the Era of Emerging Networked Society. The European Union Agency for Network and Information Security (ENISA), 2012.

[32] Bo Han, Vijay Gopalakrishnan, Lusheng Ji, and Seungjoon Lee. Network function virtualization: Challenges and opportunities for innovations. *IEEE Communications Magazine*, 53(2):90–97, 2015.

[33] Pivotal Software Inc. RabbitMQ Documentation. https://www.rabbitmq.com/documentation.html Accessed: 20.07.2018.

[34] Intel. Open Cloud Integrity Technology (Open CIT). https://01.org/opencit Accessed 28.05.2018.

[35] Ludovic Jacquin, Antonio Lioy, Diego R Lopez, Adrian L Shaw, and Tao Su. The trust problem in modern network infrastructures. In *Cyber Security and Privacy Forum*, pages 116–127. Springer, 2015.

[36] Yogesh V Jilhawar and M Emmanuel. Trustworthy Resource Scheduling using Openstack in Cloud.

[37] Imran Khan, Habib-ur Rehman, and Zahid Anwar. Design and deployment of a trusted eucalyptus cloud. In *Cloud Computing (CLOUD), 2011 IEEE International Conference on*, pages 380–387. IEEE, 2011.

[38] Doowon Kim, Bum Jun Kwon, and Tudor Dumitraş. Certified Malware: Measuring Breaches of Trust in the Windows Code-Signing PKI. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1435–1448. ACM, 2017.

[39] Doowon Kim, Bum Jun Kwon, Kristián Kozák, Christopher Gates, and Tudor Dumitraș. The Broken Shield: Measuring Revocation Effectiveness in the Windows Code-Signing PKI. In *27th {USENIX} Security Symposium ({USENIX} Security 18)*. USENIX Association, 2018.

[40] Kristián Kozák, Bum Jun Kwon, Doowon Kim, Christopher Gates, and Tudor Dumitraş. Issued for Abuse: Measuring the Underground Trade in Code Signing Certificate. *arXiv preprint arXiv:1803.02931*, 2018.

[41] Shankar Lal, Aapo Kalliola, Ian Oliver, Kimmo Ahola, and Tarik Taleb. Securing VNF communication in NFVI. In *Standards for Communications and Networking (CSCN), 2017 IEEE Conference on*, pages 187–192. IEEE, 2017.

[42] Shankar Lal, Sowmya Ravidas, Ian Oliver, and Tarik Taleb. Assuring virtual network function image integrity and host sealing in Telco cloue. In *Communications (ICC), 2017 IEEE International Conference on*, pages 1–6. IEEE, 2017.

[43] Linux. An Overview of The Linux Integrity Subsystem. http://downloads.sf.net/project/linux-ima/linux-ima/Integrity_overview.pdf Accessed 28.05.2018.

[44] Fang Liu, Jin Tong, Jian Mao, Robert Bohn, John Messina, Lee Badger, and Dawn Leaf. Nist cloud computing reference architecture. *NIST special publication*, 500(2011):1–28, 2011.

[45] Yang Luo, Wu Luo, Tian Puyang, Qingni Shen, Anbang Ruan, and Zhonghai Wu. Openstack security modules: A least-invasive access control framework for the cloud. In *Cloud Computing (CLOUD), 2016 IEEE 9th International Conference on*, pages 51–58. IEEE, 2016.

[46] Wenbo Mao. Method and apparatus for securing the full lifecycle of a virtual machine, March 7 2013. US Patent App. 13/601,053.

[47] Frank Mayer, David Caplan, and Karl MacMillan. *SELinux by example: using security enhanced Linux*. Pearson Education, 2006.

[48] Peter Membrey, Keith CC Chan, Canh Ngo, Yuri Demchenko, and Cees de Laat. Trusted virtual infrastructure bootstrapping for on demand services. In *Availability, Reliability and Security (ARES), 2012 Seventh International Conference on*, pages 350–357. IEEE, 2012.

[49] Microsoft. BitLocker. https://docs.microsoft.com/en-us/windows/security/information-protection/bitlocker/bitlocker-overview Accessed 28.05.2018.

[50] Microsoft. Secure Boot Overview. https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-8.1-and-8/hh824987(v=win.10) Accessed 28.05.2018.

[51] Yong-Hyuk Moon and Yong-Sung Jeon. Cooperative remote attestation for IoT swarms. In *Information and Communication Technology Convergence (ICTC), 2016 International Conference on*, pages 1233–1235. IEEE, 2016.

[52] Canh Ngo, Peter Membrey, Yuri Demchenko, and Cees de Laat. Security framework for virtualised infrastructure services provisioned on-demand. In *Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on*, pages 698–704. IEEE, 2011.

[53] Ian Oliver, Gabriela Limonta, and Borger Vigmostad. Towards Combining Remote Attestation and Root Cause Analysis. In *11th International Conference on the Quality of Information and Communications Technology - QUATIC 2018, Coimbra, Portugal*, September 2018.

[54] Ian Oliver, Gabriela Limonta, Borger Vigmostad, Aapo Kalliola, Yoan Miche, Silke Holtmanns, and Kiti Muller. A Testbed for Trusted Telecommunications Systems in a Safety Critical Environment. In *13th International ERCIM/EWIC-S/ARTEMIS Workshop on Dependable Smart Embedded and Cyber-physical Systems and Systems-of-Systems - DECSoS 2018, Västerås, Sweden*, September 2018.

[55] OPNFV. Transforming Networks Through Open Source NFV. 2017. https://www.opnfv.org/wp-content/uploads/sites/12/2018/02/OPNFV_AnnualReport_2017_FINAL.pdf.

[56] Justin D Osborn and David C Challener. Trusted Platform Module evolution. *Johns Hopkins APL Technical Digest (Applied Physics Laboratory)*, 32(2):536–543, 2013.

[57] Siani Pearson and Boris Balacheff. *Trusted computing platforms: TCPA technology in context*. Prentice Hall Professional, 2003.

[58] Himanshu Raj, Stefan Saroiu, Alec Wolman, Ronald Aigner, Jeremiah Cox, Paul England, Chris Fenner, Kinshuman Kinshumann, Jork Loeser, Dennis Mattoon, et al. fTPM: A Software-Only Implementation of a TPM Chip. In *USENIX Security Symposium*, pages 841–856, 2016.

[59] Sowmya Ravidas. Incorporating Trust in Network Function Virtualization. G2 pro gradu, diplomityö, 2016-10-27.

[60] Sowmya Ravidas, Shankar Lal, Ian Oliver, and Leo Hippelainen. Incorporating trust in NFV: Addressing the challenges. In *Innovations in Clouds, Internet and Networks (ICIN), 2017 20th Conference on*, pages 87–91. IEEE, 2017.

[61] Bhaskar Prasad Rimal and Ian Lumb. The Rise of Cloud Computing in the Era of Emerging Networked Society. In *Cloud Computing*, pages 3–25. Springer, 2017.

[62] Sasko Ristov, Marjan Gusev, and Aleksandar Donevski. Openstack cloud security vulnerabilities from inside and outside. *Cloud Computing*, pages 101–107, 2013.

[63] Sasko Ristov, Marjan Gusev, and Aleksandar Donevski. Security vulnerability assessment of openstack cloud. In *Computational Intelligence, Communication Systems and Networks (CICSyN), 2014 Sixth International Conference on*, pages 95–100. IEEE, 2014.

[64] Steffen Schulz, André Schaller, Florian Kohnhäuser, and Stefan Katzenbeisser. Boot Attestation: Secure Remote Reporting with Off-The-Shelf IoT Sensors. In *European Symposium on Research in Computer Security*, pages 437–455. Springer, 2017.

[65] SELinux Wiki. Main Page — SELinux Wiki. 2017. https://selinuxproject.org/w/?title=Main_Page&oldid=1842 Accessed 14.08.2018.

[66] Jyoti Shetty, MR Anala, and G Shobha. A survey on techniques of secure live migration of virtual machine. *International Journal of Computer Applications*, 39(12):34–39, 2012.

[67] Red Hat Software. A Guide to Securing Red Hat Enterprise Linux. https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/6/html/security_guide/ Accessed 14.08.2018.

[68] Tarik Taleb. Toward carrier cloud: Potential, challenges, and solutions. *IEEE Wireless Communications*, 21(3):80–91, 2014.

[69] Allan Tomlinson. Introduction to the TPM. In *Smart Cards, Tokens, Security and Applications*, pages 173–191. Springer, 2017.

[70] Trusted Computing Group. TCG PC Client Specific Implementation Specification for Conventional BIOS. 2016. https://www.trustedcomputinggroup.org/wp-content/uploads/TCG_PCClientImplementation_1-21_1_00.pdf.

[71] Wei Yang and Carol Fung. A survey on security in network functions virtualization. In *NetSoft Conference and Workshops (NetSoft), 2016 IEEE*, pages 15–19. IEEE, 2016.

[72] Fengzhe Zhang, Jin Chen, Haibo Chen, and Binyu Zang. CloudVisor: retrofitting protection of virtual machines in multi-tenant cloud with nested virtualization. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, pages 203–216. ACM, 2011.

[73] Xinwen Zhang and Jean-Pierre Seifert. Method and system for enforcing trusted computing policies in a hypervisor security module architecture, July 10 2012. US Patent 8,220,029.

A Hardware Specification

This Appendix shows the hardware used for the OpenStack implementation in this thesis. Hardware details are shown in Tables A1, A2 and A3.

Name	AirFrame 1 (A1)
Processor	Intel(R) Xeon(R) CPU E5-2630 v3 @ 2.40GHz - 31 Cores
System Memory	62GB
Hard Drive	1.8TB LVM
Operating System	Ubuntu 17.10 with kernel 4.10.0-42-generic
OpenStack version	Pike
OpenStack roles	Controller and compute
TPM version	2.0

Table A1: AirFrame 1 hardware specification

Name	AirFrame 2 (A2)
Processor	Intel(R) Xeon(R) CPU E5-2630 v4 @ 2.20GHz - 32 cores
System Memory	62GB
Hard Drive	1.5TB LVM
Operating System	Ubuntu 17.10 with kernel 4.10.0-42-generic
OpenStack version	Pike
OpenStack role	Compute
TPM version	2.0

Table A2: AirFrame 2 hardware specification

Name	AirFrame 3 (A3)
Processor	Intel(R) Xeon(R) CPU E5-2680 v4 @ 2.40GHz - 53 cores
System Memory	125GB
Hard Drive	1.8TB LVM
Operating System	Ubuntu 17.10 with kernel 4.10.0-42-generic
OpenStack version	Pike
OpenStack role	Compute
TPM version	2.0

Table A3: AirFrame 3 hardware specification

B OpenStack Details

This Appendix contains the full state diagram for VNF instances and a list of the OpenStack configuration files.

B.1 VNF State Diagram

VNF instances can perform many operations that change the state of the VNF. A full state diagram is shown in Figure B1.

B.2 Configuration Files

An OpenStack installation is configured through a set of configuration files. This sections lists the configuration files for the OpenStack services Glance, Keystone, Neutron, Horizon and Nova.

```
/etc/glance/glance-api.conf
/etc/glance/glance-cache.conf
/etc/glance/glance-registry.conf
/etc/glance/glance-scrubber.conf
/etc/glance/glance-registry-paste.ini
/etc/glance/glance-api-paste.ini
/etc/keystone/keystone.conf
/etc/keystone/keystone-paste.ini
/etc/neutron/neutron.conf
/etc/neutron/dhcp_agent.ini
/etc/neutron/l3_agent.ini
/etc/neutron/api-paste.ini
/etc/neutron/metadata_agent.ini
/etc/neutron/plugins/evs/evs_plugin.ini
/etc/openstack_dashboard/local_settings.py
/etc/apache2/2.4/httpd.conf
/etc/apache2/2.4/conf.d/openstack-dashboard-http.conf
/etc/nova/nova.conf
/etc/nova/api_paste.ini
```

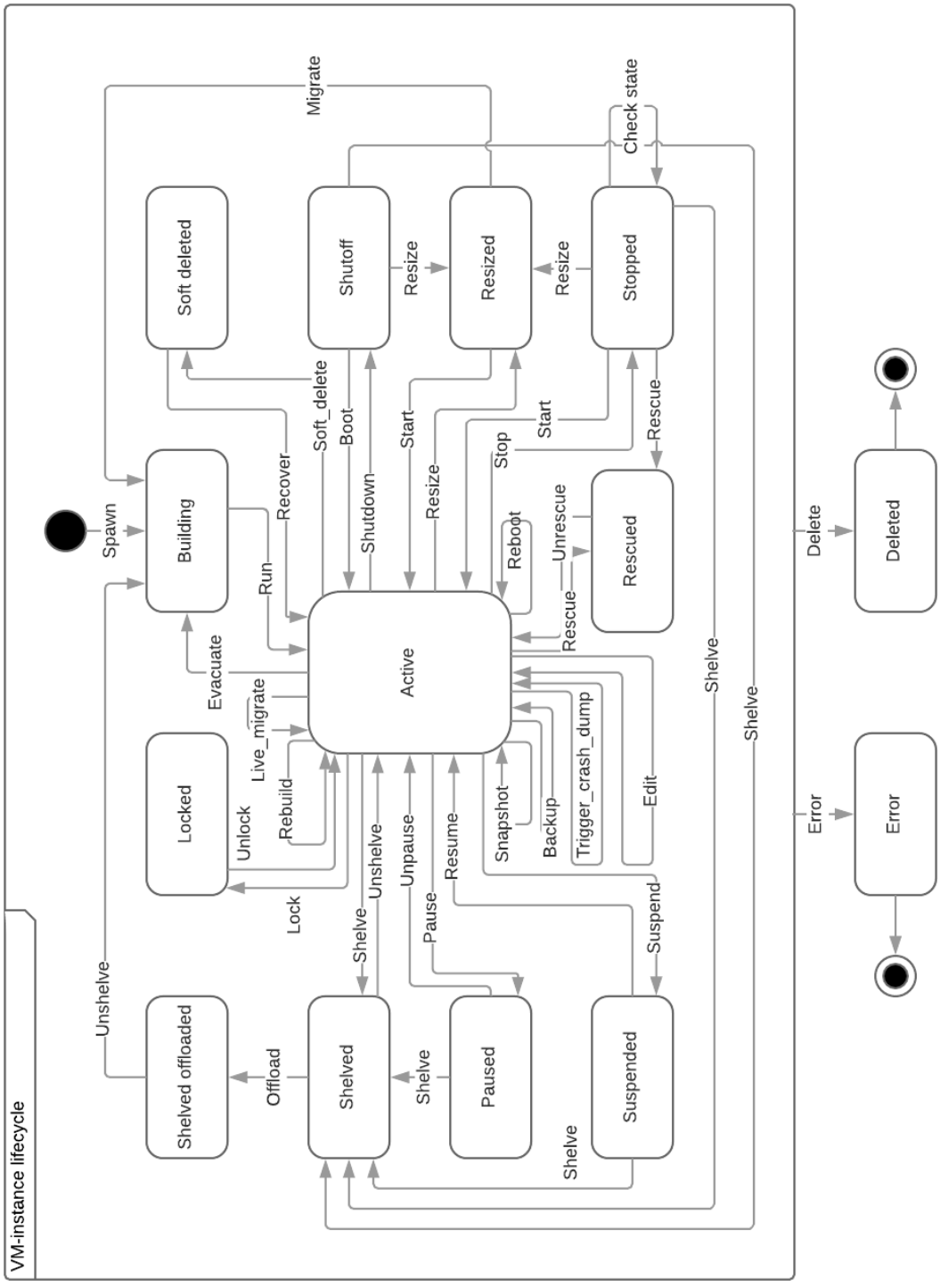

Figure B1: Full OpenStack VM state diagram

C Enabling Trust

To enable trust in a system many software solutions need to work together. This Appendix shows how to use tboot, Linux IMA and SELinux for secure system measurements.

C.1 Tboot

These steps enable tboot[25] for a server with TPM 2.0 running Ubuntu with Linux-4.10.0-42 kernel. Unless otherwise defined, tboot fills the DRTM in PCRs 17 and 18.

- Enable TPM in BIOS then reboot.
- Enable Intel TXT, SMX and VMX in BIOS then reboot.
- Check that TPM drivers are working and that the TPM can be found with `$ ls /dev/ | grep tpm`.
- Install tboot with `$ apt install tboot`.
- Update grub with `$ update-grub` or `$ update-grub2` depending on local grub version.
- Reboot computer and choose tboot in boot menu. This can be made automatic by updating `/etc/default/grub` and reloading grub.
- Check for valid boot with`txt-stat | grep secret`. This should return `secrets: TRUE`.

C.2 Linux IMA

Linux IMA[26] is part of the Linux kernel. Many recent Linux distributions have it included; however, if it is not included the kernel would have to be rebuilt with Linux IMA to be able to use it. The following steps enables Linux IMA in Ubuntu with Linux-4.10.0-42 kernel.

- Check that IMA exists in securityfs with `$ ls /sys/kernel/security/ | grep ima`.

[25] https://sourceforge.net/projects/tboot/
[26] https://sourceforge.net/projects/linux-ima/

  - If IMA does not exist, upgrade to a newer kernel or rebuild current kernel with IMA enabled.

- Check that securityfs is mounted with `$ mount | grep securityfs`.
  - If it is not mounted, mount with `$ mount -t securityfs securityfs /sys/kernel/security`.
- Define a policy for what to measure. The default policy can be found at `/etc/ima/ima-policy`.
- Add one or more IMA policies to grub. To enable TCB default policy, add `ima_policy=tcb` to `GRUB_CMDLINE_LINUX_DEFAULT` in `/etc/default/grub`.
- Reload grub and reboot to enable grub changes.

C.3 SELinux Tags for Linux IMA Measurements

This Section presents the process of measuring the main OpenStack Nova configuration file, which is stored in `/etc/nova/nova.conf`. This measurement will be written by Linux IMA and stored in TPM PCR 14. A full list of OpenStack configuration files can be found in Appendix B Section B.2.

This process depends on SELinux and Linux IMA being enabled in the Linux kernel. The process of enabling Linux IMA is described in Section C.2. The following steps measures `/etc/nova/nova.conf` by marking it using SELinux types and adding the SELinux tag to the Linux IMA policy.

- Create a `openstack_measure` module.
  - `$ mkdir /usr/share/selinux/packages/openstack_measure`
  - Make file `/usr/share/selinux/packages/openstack_measure/openstack_measure.te` with the content below.

    ```
    policy_module(openstack_measure, 1.0)
    type openstack_measure_t;
    ```

- Build module with `$ make -f /usr/share/selinux/devel/Makefile`
- Load module with `$ semodule -i openstack_measure.pp`
- Tag `/etc/nova/nova.conf` with `openstack_measure_t` using `$ chcon -v --type=openstack_measure_t /etc/nova/nova.conf`

- Tell IMA to measure all files marked with `openstack_measure_t` in PCR 14 by adding `measure func=FILE_CHECK obj_type=openstack_measure_t pcr=14` to the default IMA policy in `/etc/ima/ima-policy`.
- Reboot to enable IMA measurements.

D Attestation Server Details

This Appendix lists the attestation server REST API and presents example attestation server entries and events.

D.1 REST API

This Section has the REST API for the attestation server block added to the NFV architecture, which can be seen in Table D1.

Endpoint	Methods	Description
`/element/tpm`	POST	Add TPM element.
`/element/vnf_image`	POST	Add VNF image.
`/element/vnf_instance`	POST	Add VNF instance.
`/element/<database_id>`	GET | DELETE | PUT	Get or modify an element.
`/element/openstack/<openstack_id>`	GET | DELETE | PUT	Get or modify an element.
`/event`	POST	Add event.
`/policy`	POST	Add policy.
`/policy/<database_id>`	GET | DELETE | PUT	Get or modify a policy.
`/tpm_quote`	POST	Add TPM quote.
`/tpm_quote/<database_id>`	GET	Get a TPM quote.
`/record`	POST	Add record.
`/record/<database_id>`	GET | DELETE | PUT	Get or modify a record.

Table D1: Attestation server REST API

D.2 Example Database Entries

This Section has example values for database entries implemented in this thesis.

```
{
  "_id": {
    "$oid": "5b4894d419af5157a0f61260"
  },
  "ak": "-----BEGIN PUBLIC KEY-----
  MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAmZtDWjzQFuZPi
  qkDLOJhF3kWsu0pXT1zzjwDkbMBYtVnZB99c4w8afG5hQWNswqaDg/tik
  B55vJ7tS94tPM8T/CvWt0qLoR0z8Pg1o+V2WcTJEnqYi/X9Rs0e9jNNRz
  rp40LquRR6BCJIwt9tDvW\n4GnU2AEDpRDodUGYUsN4tN84sYLesDKCZc
  'VjEInxBidWoA4CUPdJ3NexZAIgYRFsXgi3joPTYne2ySKhKpTV8g7rkO
  9Jjkfd7EE0OvvPx4aQ2ke0tWBGDi+HwTrBMOzNRKo5mNnS32H9cl0yeaC
  6qkAio2LvbwhJMELlAAOmgmAAEc4P0fSwgqZJDkXfhUmmwIDAQAB
  -----END PUBLIC KEY-----",
  "ek": "-----BEGIN PUBLIC KEY-----
```

```
  MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAh6V2QY/i15sdH
  j+U4Y9r\nWSUU6jejZPScL+8K0LWrCxQmEycLGTNMwXw4f0TabeIKPpY2
  BcmWRKWJ7c2a/b4vp7exqgYGBru0B//qdrZvcth1OKEZUsrIFYGvORjUf
  aSAoMiB4LQkqpEf/1jxO/zI6EanTHm1oN/qqHh2AkKHJUJ2rQUobwE+2w
  xEmvnW7AVD67xliBfGczC9LDbwDOagDalbLRLeiFt1qTj23tDhJGjtYqK
  vLMBTp4mzBukAqnuEEU+GwX8/Bdy7NPGQE/GVXvfB0q6XHHgn8EQSNTNs
  zPUwMhPRgbL3mJ8NWoQ+BuNSdnifC7+M3WCfI0TVUePb/QIDAQAB
  −−−−−END PUBLIC KEY−−−−−",
  "ip": "10.144.104.19",
  "kinds": [
    "NFVIElement::TpmMachine",
    "NFVIElement::Machine"
  ],
  "policy_list": [
        "5b4da33c352c656e9247a043",
        "5b3f081b19af5170379043a0"
  ],
  "name": "compute",
  "openstack_id": "4d4542b2−37a6−44fe−9c84−45bd750b5b90",
  "status": true,
  "timestamp": {
    "$date": 1533022285447
  },
  "uname": "Linux compute 4.10.0−42−generic #46−Ubuntu SMP Mon Dec 4 14:38:01
      UTC 2017 x86_64 x86_64 x86_64 GNU/Linux\n",
  "signature": "eqh7JzKIdXDpdMdJsjEic7XEku43bJmYm/nKPu++xJn+
  Zj7ijGZ6HbHkBsyodoUfSrfpcMYgZwu2pMNLcAancw/2y+RBSa66s3x9nX
  1I3lfU0wAWinp43xh9d/Jkux9H4cWLMb5IWYXuJsTJ6CDpEBdjME/xIx4M
  L13S8K95vu3x4/ul7A4eul2NDkJ6tok7jZ1NWGCyL5pB2znHT4RHkLwU1i
  sTVRN 8iJxUh6mrU2ycBUop09eS3IQhB4Jm92BQR2deiqNZWnbFrPXeqX
  NX/X9IAnVgQEca1dHI5tD7HUioSNnUncaTe+ZIctoi8P4yN8j0z4EaUmBe
  a2qjUF7KeB7XdWrHTXjfaorqwJbKu85QHsQwmsoSHr2Yc4pYXEuANoAfjO
  xGJABR4rn6949TRQh+puXqXHrmVqaiLo9heZAtzckrUANWGPdBAfTDD7ka
  ryuXZP4rfLsMSPC65kWxOT9JU4dVjCp7mO4/nvEFGdrEybDPUNALKkp4uP
  CxJfU+ZM9zu+EV7d1oRvokXT22QBPfcRAKQhXofnMLYUhwSjNUqNCjzP+h
  Bg0RlySyw//IzoGqhJ2C1A4pNcZ+2QVqn27qeb/Ng2ZNTXhcOKyycA3tma
  oTLpTxuH26akXFMudma6Zli1s6xLkxBJW+hT+nM5kRkFoSEfVJigBuRVEi
  /ZE="
}
```


Listing 10: Example TPM element entry


```
{
  "_id": {
    "$oid": "5b4d9b71352c656e92479f19"
  },
  "expected_value": "Sl7gGpD2R7mcGNzKwd/6huDsspWsJ4GwpNZbe+3M1Yw=",
```

```
  "kind": "Policy::TPM2.0",
  "name": "compute's CRTM (SHA1)",
  "pcrs": "sha1:0",
  "timestamp": {
    "$date": 1531812721688
  },
  "signature": "eqh7JzKIdXDpdMdJsjEic7XEku43bJmYm/nKPu++xJn+
  Zj7ijGZ6HbHkBsyodoUfSrfpcMYgZwu2pMNLcAancw/2y+RBSa66s3x9nX
  1I3lfU0wAWinp43xh9d/Jkux9H4cWLMb5IWYXuJsTJ6CDpEBdjME/xIx4M
  L13S8K95vu3x4/ul7A4eul2NDkJ6tok7jZ1NWGCyL5pB2znHT4RHkLwU1i
  sTVRN 8iJxUh6mrU2ycBUop09eS3IQhB4Jm92BQR2deiqNZWnbFrPXeqX
  NX/X9IAnVgQEca1dHI5tD7HUioSNnUncaTe+ZIctoi8P4yN8j0z4EaUmBe
  a2qjUF7KeB7XdWrHTXjfaorqwJbKu85QHsQwmsoSHr2Yc4pYXEuANoAfjO
  xGJABR4rn6949TRQh+puXqXHrmVqaiLo9heZAtzckrUANWGPdBAfTDD7ka
  ryuXZP4rfLsMSPC65kWxOT9JU4dVjCp7mO4/nvEFGdrEybDPUNALKkp4uP
  CxJfU+ZM9zu+EV7d1oRvokXT22QBPfcRAKQhXofnMLYUhwSjNUqNCjzP+h
  Bg0RlySyw//IzoGqhJ2C1A4pNcZ+2QVqn27qeb/Ng2ZNTXhcOKyycA3tma
  oTLpTxuH26akXFMudma6Zli1s6xLkxBJW+hT+nM5kRkFoSEfVJigBuRVEi
  /ZE="
}
```


Listing 11: Example policy entry


```
{
  "_id": {
    "$oid": "5b5af087352c6520115e5300"
  },
  "hash_list": [
    {
      "hash": "7c921ce7b7696c80d5e04914a8de5d3512cb48dedacde
      f79434588d9c1570701",
      "hash_type": "sha256"
    },
    {
      "hash": "e924d1602ff88edca0a02e2ff129a810",
      "hash_type": "md5"
    },
    {
      "hash": "cb6f44d6986d407fbb2be638f1f44c67d1562d42",
      "hash_type": "sha1"
    }
  ],
  "kinds": [
    "VNFElement::VNFImage"
  ],
  "name": "Ubuntu_trusted",
  "openstack_id": "dcb8967e-890a-43ab-bf49-55893575fe34",
```

```
  "status": true,
  "timestamp": {
    "$date": 1532689593964
  },
  "signature": "eqh7JzKIdXDpdMdJsjEic7XEku43bJmYm/nKPu++xJn+
  Zj7ijGZ6HbHkBsyodoUfSrfpcMYgZwu2pMNLcAancw/2y+RBSa66s3x9nX
  1I3lfU0wAWinp43xh9d/Jkux9H4cWLMb5IWYXuJsTJ6CDpEBdjME/xIx4M
  L13S8K95vu3x4/ul7A4eul2NDkJ6tok7jZ1NWGCyL5pB2znHT4RHkLwU1i
  sTVRN 8iJxUh6mrU2ycBUop09eS3IQhB4Jm92BQR2deiqNZWnbFrPXeqX
  NX/X9IAnVgQEca1dHI5tD7HUioSNnUncaTe+ZIctoi8P4yN8j0z4EaUmBe
  a2qjUF7KeB7XdWrHTXjfaorqwJbKu85QHsQwmsoSHr2Yc4pYXEuANoAfjO
  xGJABR4rn6949TRQh+puXqXHrmVqaiLo9heZAtzckrUANWGPdBAfTDD7ka
  ryuXZP4rfLsMSPC65kWxOT9JU4dVjCp7mO4/nvEFGdrEybDPUNALKkp4uP
  CxJfU+ZM9zu+EV7d1oRvokXT22QBPfcRAKQhXofnMLYUhwSjNUqNCjzP+h
  Bg0RlySyw//IzoGqhJ2C1A4pNcZ+2QVqn27qeb/Ng2ZNTXhcOKyycA3tma
  oTLpTxuH26akXFMudma6Zli1s6xLkxBJW+hT+nM5kRkFoSEfVJigBuRVEi
  /ZE="
}
```


Listing 12: Example VNF image entry


```
{
  "_id": {
    "$oid": "5b6c0ffa352c65376cbc9fa2"
  },
  "element_id": "5b4894d419af5157a0f61260",
  "kind": "Quote::TPM2.0",
  "pcrs": {
    "sha1": {
      "0": "ae62d40b59561f52419a6dbb1fe3f589e7c67856",
      "1": "b2a83b0ebf2f8374299a5b2bdfc31ea955ad7236",
      "2": "b2a83b0ebf2f8374299a5b2bdfc31ea955ad7236",
      "3": "b2a83b0ebf2f8374299a5b2bdfc31ea955ad7236",
      "4": "b2a83b0ebf2f8374299a5b2bdfc31ea955ad7236",
      "5": "870a58e79a43e9dcdd273b6efa9975d882ce7420",
      "6": "b2a83b0ebf2f8374299a5b2bdfc31ea955ad7236",
      "7": "4037336fa7bc0eabe3778fcfff5fcd0ee6adcde3",
      "8": "0000000000000000000000000000000000000000",
      "9": "0000000000000000000000000000000000000000",
      "10": "144851a7c93d96ceff3ab81eaa75ae021deb350f",
      "11": "0000000000000000000000000000000000000000",
      "12": "0000000000000000000000000000000000000000",
      "13": "0000000000000000000000000000000000000000",
      "14": "0000000000000000000000000000000000000000",
      "15": "0000000000000000000000000000000000000000",
      "16": "f0e5a9f7d5a33a281fff32e565cb407df5b191c1",
      "17": "5ffa409f3ddbb8b339d9e7e9fbcec4a38d6dc817",
```

```
        "18": "9b8547051d0e68839490e82f99a17d78e3e94fed",
        "19": "0000000000000000000000000000000000000000",
        "20": "0000000000000000000000000000000000000000",
        "21": "          ffffffffffffffffffffffffffffffff          ",
        "22": "          ffffffffffffffffffffffffffffffff          ",
        "23": "0000000000000000000000000000000000000000"
      },
      "sha256": {
"0":" d8a76f44656e5b7ed75ddc6c19071d8594e99edb67c54c0f5f562a8bdaa26bbf",
"1":"3 d458cfe55cc03ea1f443f1562beec8df51c75e14a9fcf9a7234a13f198e7969",
"2":"3 d458cfe55cc03ea1f443f1562beec8df51c75e14a9fcf9a7234a13f198e7969",
"3":"3 d458cfe55cc03ea1f443f1562beec8df51c75e14a9fcf9a7234a13f198e7969",
"4":"3 d458cfe55cc03ea1f443f1562beec8df51c75e14a9fcf9a7234a13f198e7969",
"5":" c38c9900b691c6f2d8b5c4959e3548a14b7ba61713a194560c545c62c4b0b4a5",
"6":"3 d458cfe55cc03ea1f443f1562beec8df51c75e14a9fcf9a7234a13f198e7969",
"7":" b5710bf57d25623e4019027da116821fa99f5c81e9e38b87671cc574f9281439",
"8":"0000000000000000000000000000000000000000000000000000000000000000",
"9":"0000000000000000000000000000000000000000000000000000000000000000",
"10":"0000000000000000000000000000000000000000000000000000000000000000",
"11":"0000000000000000000000000000000000000000000000000000000000000000",
"12":"0000000000000000000000000000000000000000000000000000000000000000",
"13":"0000000000000000000000000000000000000000000000000000000000000000",
"14":"0000000000000000000000000000000000000000000000000000000000000000",
"15":"0000000000000000000000000000000000000000000000000000000000000000",
"16":"4 ca16202817eb69aba12b94c1410ba17f0d136879377bf9aeb62a117597870e2",
"17":"13271ef5faf9c806e07d33ca81837b7bb8bf2e112e72b7ee3d8e8fe0b8bb784e",
"18":"3 d403e3eb24aa3a73a766f9782ab728bc9a94c5c11dab7734729aff2fdba2336",
"19":"0000000000000000000000000000000000000000000000000000000000000000",
"20":"0000000000000000000000000000000000000000000000000000000000000000",
"21":"          ffffffffffffffffffffffffffffffff          ",
"22":"          ffffffffffffffffffffffffffffffff          ",
"23":"0000000000000000000000000000000000000000000000000000000000000000"
      }
    },
    "policy_id": "5b4d9b71352c656e92479f19",
    "quote": {
      "attested": "Sl7gGpD2R7mcGNzKwd/6huDsspWsJ4GwpNZbe+3M1Yw=",
      "clock": 3497869051,
      "extraData": "8RrHO3pRTf+bFGPFmkso4Q==",
      "firmwareVersion": 1407374883832066,
      "magic": "/1RDRw==",
      "qualifiedSigner": "MXLdgzlTqCViuUynaGjeP3rXmiQyW0q1J6NWxxrABSY=",
      "quoteFile": "/1
          RDR4AYACIACzFy3YM5U6glYrlMp2ho3j9615okMltKtSejVscawAUmABDxGs
      c7elFN/5sUY8WaSyjhAAAAANB9PvsAAADVAAAAAQAABQAAAARBAgAAAA
      EABAMBAAAAIEpe4BqQ9ke5nBjcysHf+obg7LKVrCeBsKTWW3vtzNWM",
      "resetCount": 213,
      "restartCount": 1,
```

```
    "safe": 0,
    "signature": "ABQACwEAFoJb2HmsqGNkLv7+0IuvShY1c7Pt/wOf8N666OVW
    cIbX5PzaUi1s6kdOGUKdnHzqnIFI2C6L+UWRE9EJVAX6lTPJbKlW3AWljfLAmA
    bCwOZVJaeUplU+ClTWWnvxq7zlvXIdXWESRPOodd93qJ+2+Xhc89lMslZUudr2
    USVVvH+C6hgjxqX8XQtk9gJ0/UjIxG8pNKYIvBlDHjRud/FbNs3yltS0/hHtDs
    Gg06LIDie/a0FiDOjTQ66OJHqBfUVCW5JZUWAAz2otzdWZ/nUy7EDbAaqrqLgE
    uytyKrbbV/5KngJlBsbPS5JqUBE808bEbp5zX1iTmqzashytrmCMTA==",
    "type": "gBg="
  },
  "timestamp": {
    "$date": 1533808634631
  }
```


Listing 13: Example quote entry


```
{
  "VNF_image": "85a7cf0b-cd2f-429a-b3c3-44c88b923e4e",
  "_id": {
    "$oid": "5b6c4074352c65376cbc9faa"
  },
  "kinds": [
    "VNFElement::VNFInstance"
  ],
  "migration_record": null,
  "name": "test-vnf",
  "openstack_id": "52de4f85-e12f-48cb-b958-9fd6df080737",
  "previous_suspend_hash_list": [],
  "running_on": "compute",
  "state": "Request_to_start",
  "status": true,
  "timestamp": {
    "$date": 1533821044910
  },
  "signature": "ABQACwEAGe9IekxuQcCX9MG7b+ztFNCmJYxC48Nb4AW8n/10Oh
  6+ycnfauqKairv5A1/gS2sintZWSr06s/x4RUCKV2NC5Ja6Qnk+UqetQlpLN3F5T
  ZRixgzX5yVamBUD9LvuJJiZsFfmQqQscKco3GsPQMTbklKAbIHS/vJmbK4jHE2d6
  YURa+RG3qsa4dPESZcykItxt5eM8nYygflc9qp81JIJUHnSCycHaNwgErE3N0DFc
  VIHESz2X2k+NDKy9KBwxb7Q91sg8S0pLZYAPAWvvKmUz0AAn++
      ueAcl5QY52dBQt
  PUgAETRiYuI/75OcPbsEJbcRMp74ZK2UWFM/TWUVAJWw=="
}
```


Listing 14: Example VNF instance entry


```
{
```

```
  "_id": {
    "$oid": "5b6c4c7f352c65376cbca045"
  },
  "completed": true,
  "from_host": "controller",
  "from_instance": "5b6c4c67352c65376cbca040",
  "kind": "Record::Migration",
  "openstack_id": "ced3eb65-8882-4fde-9fd6-3de72a93f4c5",
  "status": true,
  "timestamp": {
    "$date": 1533824134428
  },
  "to_host": "compute",
  "to_instance": "5b6c4c7f352c65376cbca044",
  "transfer_hash_list": [
    {
      "hash": "a0392d5f2bee2ca6437591c7146860d43e319f438f34eeaf298607d23fdfa6d0",
      "hash_type": "sha256"
    },
    {
      "hash": "c890180c571e1677b12319eb3fb5e623e5aaf833",
      "hash_type": "sha1"
    },
    {
      "hash": "2c10fdfa4816eda0aa3afc5e38cc38f9",
      "hash_type": "md5"
    }
  ],
  "signature": "ABQACwEAGe9IekxuQcCX9MG7b+ztFNCmJYxC48Nb4AW8n/10Oh
  6+ycnfauqKairv5A1/gS2sintZWSr06s/x4RUCKV2NC5Ja6Qnk+UqetQlpLN3F5T
  ZRixgzX5yVamBUD9LvuJJiZsFfmQqQscKco3GsPQMTbklKAbIHS/vJmbK4jHE2d6
  YURa+RG3qsa4dPESZcykItxt5eM8nYygflc9qp81JIJUHnSCycHaNwgErE3N0DFc
  VIHESz2X2k+NDKy9KBwxb7Q91sg8S0pLZYAPAWvvKmUz0AAn++
      ueAcl5QY52dBQt
  PUgAETRiYuI/75OcPbsEJbcRMp74ZK2UWFM/TWUVAJWw=="
}
```


Listing 15: Example migration record entry

D.3 Example Event Entries

This Section lists example events for OpenStack scheduler filter and VNF suspension element update.


```
{
  "_id": {
```

```
    "$oid": "5b62d836352c65376cbc9ec4"
  },
  "event_type": "OpenStack filter",
  "hypervisor_name": "compute",
  "hypervisor_openstack_id": "4d4542b2-37a6-44fe-9c84-45bd750b5b90",
  "nonce": "7ad72cea-9a26-4924-aeaf-f27997d3b08e",
  "result": true,
  "result_reasons": [
    "Trust level set to True",
    "Hypervisor found in attestation server",
    "Hypervisor and instance are both trusted",
    "Image found in attestation server",
    "Expected sha256 has was
        e137062a4dfbb4c225971b67781bc52183d14517170e16a3841d16f962ae7470.\n
        Measured hash was
        e137062a4dfbb4c225971b67781bc52183d14517170e16a3841d16f962ae7470.",
    "Expected md5 has was f8ab98ff5e73ebab884d80c9dc9c7290.\n Measured hash was
        f8ab98ff5e73ebab884d80c9dc9c7290.",
    "Expected sha1 has was 615ca705b98c24bf4ccb535ab3e0611486b17c2a.\n Measured
        hash was 615ca705b98c24bf4ccb535ab3e0611486b17c2a.",
    "VNF image passed hash comparisons"
  ],
  "timestamp": {
    "$date": 1533204534825
  },
  "vnf_image_openstack_id": "85a7cf0b-cd2f-429a-b3c3-44c88b923e4e",
  "vnf_instance_openstack_id": "6ddbaad8-4c7e-4aa7-bdf5-f87c1a5529c9"
}
```


Listing 16: Event for OpenStack scheduler filter


```
{
  "_id": {
    "$oid": "5b6c440c352c65376cbc9fb1"
  },
  "element_id": "5b6c43eb352c65376cbc9fae",
  "event_type": "Element update",
  "new_element": {
    "VNF_image": "85a7cf0b-cd2f-429a-b3c3-44c88b923e4e",
    "_id": "5b6c43eb352c65376cbc9fae",
    "kinds": [
      "VNFElement::VNFInstance"
    ],
    "migration_record": null,
    "name": "test-vnf",
    "openstack_id": "dc510117-475a-44f2-8ed9-ba21c592e416",
    "previous_suspend_hash_list": [
```

```
      {
        "hash": "bc12d53ee991ff1a745c867fe0a25ce9551debcfec639fd4fe5de24ed266c28c",
        "hash_type": "sha256"
      },
      {
        "hash": "a897948b229e1b882ac093a552b3591075ae19cf",
        "hash_type": "sha1"
      },
      {
        "hash": "884a07777dc50457126e9ec86b4266a8",
        "hash_type": "md5"
      }
    ],
    "running_on": "compute",
    "state": "suspended",
    "status": true,
    "timestamp": "2018-08-09 13:39:24.032000"
  },
  "new_parameters": {
    "previous_suspend_hash_list": [
      {
        "hash": "bc12d53ee991ff1a745c867fe0a25ce9551debcfec639fd4fe5de24ed266c28c",
        "hash_type": "sha256"
      },
      {
        "hash": "a897948b229e1b882ac093a552b3591075ae19cf",
        "hash_type": "sha1"
      },
      {
        "hash": "884a07777dc50457126e9ec86b4266a8",
        "hash_type": "md5"
      }
    ],
    "state": "suspended"
  },
  "nonce": "ed7452fc-49a2-4ad9-88f0-b4c4d3554c1f",
  "old_element": {
    "VNF_image": "85a7cf0b-cd2f-429a-b3c3-44c88b923e4e",
    "_id": "5b6c43eb352c65376cbc9fae",
    "kinds": [
      "VNFElement::VNFInstance"
    ],
    "migration_record": null,
    "name": "test-vnf",
    "openstack_id": "dc510117-475a-44f2-8ed9-ba21c592e416",
    "previous_suspend_hash_list": [],
    "running_on": "compute",
    "state": "active",
```

```
65      "status": true,
66      "timestamp": "2018-08-09 13:39:21.295000"
67    },
68    "result": true,
69    "timestamp": {
70      "$date": 1533821964072
71    }
72  }
```

Listing 17: Event for updated suspended VNF instance